Spring 2014

DETECTING AIR TRAFFIC CONTROLLER INTERVENTIONS IN RECORDED AIR TRANSPORTATION SYSTEM DATA

Yul Kwon
Purdue University

Graduate School ETD Form 9
(Revised 01/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Yul Kwon

Entitled
DETECTING AIR TRAFFIC CONTROLLER INTERVENTIONS IN RECORDED AIR TRANSPORTATION SYSTEM DATA

For the degree of Master of Science in Industrial Engineering

Is approved by the final examining committee:

Steven J. Landry

Hong Wan

Karen Marais

To the best of my knowledge and as understood by the student in the *Thesis/Dissertation Agreement. Publication Delay, and Certification/Disclaimer (Graduate School Form 32)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Steven J. Landry

Approved by Major Professor(s):

Approved by: Abhi Deshmukh 04/26/2014

Head of the Department Graduate Program Date

DETECTING AIR TRAFFIC CONTROLLER INTERVENTIONS IN RECORDED AIR TRANSPORTATION SYSTEM DATA

A Thesis

Submitted to the Faculty

of

Purdue University

by

Yul Kwon

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Industrial Engineering

May 2014

Purdue University

West Lafayette, Indiana

To my loving and supporting family.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Steven Landry, for his patience, guidance, and encouragement throughout the course of my research. He has taught me to realize the importance of the pactical and the theoretical aspects of human factors research within the air transportation systems domain. I would like to further extend my appreciation to Dr. Hong Wan and Dr. Karen Marais for their willingness to serve on my committee.

Lastly and most importantly, I thank my wife and children for bringing laughter into my everyday life. Their support and encouragement have been invaluable throughout the process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Kwon, Yul M.S.I.E., Purdue University, May 2014. Detecting Air Traffic Controller Interventions in Recorded Air Transportation System Data. Major Professor: Steven J. Landry.

In this study, I propose a systematic method of detecting aircraft deviation due to air traffic controller (ATC) intervention. The aircraft deviations associated with ATC interventions are detected using a heuristic algorithm developed from analyzing the actual positions of an aircraft to its filed flight plan when the aircraft trajectories were identified as having an encounter in a loss-of-separation incident. An actual (closed-loop) flight trajectory of the Cleveland Air Route Traffic Control Center (ZOB ARTCC) was collected from the FlightAware database. This was compared with the corresponding planned (open-loop) trajectory dataset generated by the Microsoft©Flight Simulator X (FSX). I implemented a conflict-detection algorithm in Matlab to identify open-loop flight trajectories that encounters in loss-of-separation. I analyzed the differences between the closed-loop and open-loop flight trajectories of aircrafts that were identified to have encounters in loss of separation. The analysis identified operationally significant deviations in the closed-loop trajectory data with respect to the horizontal paths of the aircrafts. I then developed and validated a heuristic algorithm, the ATC intervention detection algorithm, based on the findings from the analysis. When used with a test dataset to validate the algorithm, it achieved an 85.7% detection rate in detecting horizontal deviations made by the ATC in resolving identified conflicts, and a false-alarm rate of 68%. In addition to the ATC intervention detection algorithm, I present in this paper an analysis of deviated flight trajectories in an effort to display how the presented methodology can be utilized to provide insight into air traffic controller resolution strategies.

1. INTRODUCTION

In an effort to better aid pilots and air traffic controllers under the future Air Traffic Control (ATC) system, there has been a significant investment made by government agencies and industry stakeholders in the study and development of decision support tools (DSTs) over the past two decades, especially in relation to conflict detection and resolution (CD&R) models (Mohleji & Ostwald, 2003). However, CD&R models that have been proposed or have been implemented into DSTs consider only a limited scope of conditions in a system that is very complex. The majority of the CD&R models focus on utilizing mathematical algorithms that limit the air traffic controllers' understanding of the presented resolution strategies (Gordon, Shorrock, & Pozzi, 2005; Kuchar & Yang, 2000). Having the air traffic controllers execute a resolution strategy with limited knowledge of a solution process that differs from the "mental model" they use in practice makes it difficult for the DSTs to be utilized effectively. Implementing DSTs for operational use is unlikely until the human-centered computing issues are resolved with the new system being developed (S. J. Landry, 2011). The integration of ATC knowledge will provide valuable insight into developing an advanced DST that can be accepted and trusted by users in a future operational environment. User acceptance of DSTs can be achieved by, among other things, suggesting conflict resolutions that are plausible and comprehensible for all operators in the system.

Previous studies have addressed the need for a human-centered CD&R models and identified strategies and rules of air traffic controllers based on simulations, interviews, and laboratory experiments (S. Inoue et al., 2012; Kirwan, Flynn, & Bretigny, 2001; Özgür & Cavcar, 2008; Rantanen & Wickens, 2012; Vela, Clarke, Feron, Durand, & Singhose, 2011). However, the previous work does not present a formal methodology

or framework that could systematically identify and characterize the knowledge of air traffic controllers when executing their tasks under practical operations. Also, human subject experiments are costly to conduct, therefore limiting the amount of data that can be collected with fixed monetary resources. Researchers simplifying an event in order to show clear casual links between variables make experiments artificial and results may not generalize well to the real world.

The research presented in this thesis is an initial study to develop a systematic method that can detect air traffic controller intervention under identified events using historical aircraft flight trajectory data. The capability of automatically detecting air traffic controller intervention by accessing mass amounts of historical aircraft flight trajectory data will allow us to better understand and characterize when and how air traffic controllers intervene with traffic to manage the airspace in current operations.

Even though the ultimate goal of this course of research is to characterize all the possible reasons for ATCs to intervene with traffic, the present study focuses on detecting intervention under identified loss of separation occurrences. A significant portion of air traffic controller workload comes from the process of detecting potential conflicts and implementing feasible strategies to manage air traffic situations. The analysis of these control strategies is expected to contribute toward a better understanding of the higher-level cognitive strategies of controllers. This information can provide key knowledge for establishing a model of how ATCs intervene to ensure aircraft separation, and this model can be used for the design and implementation of decision-support tools to better aid air traffic controllers in a plausible manner.

The work is organized as follows:

- In Chapter 2, I review past and present research in relation to the development of a human-centered ATC support system and in association with conflict detection and resolution models.

- In Chapter 3, I describe the procedural steps that were taken in the development of the ATC intervention detection algorithm.

- In Chapter 4, I present the results regarding the accuracy of the algorithm and an analysis of ATC control strategies.
- In Chapter 5, I provide the study conclusions and recommendations for the direction of future work.

2. LITERATURE REVIEW

The projected growth in the number of aircraft operating in the U.S. national airspace system (NAS) is likely to exceed the capability of the air traffic control system currently in operation. Consequently, the federal government and other air navigation service providers are initiating a new concept of operation, known as the Next Generation Air Transportation System (NextGen), in an effort to improve the capacity and throughput of the existing airspace. Its concept of operation will require incorporating new technologies into the existing air traffic system infrastructure, as well as implementing new operational concepts designed to accommodate the increase in air traffic capacity while maintaining the current safety and security level (Erzberger & Paielli, 2002; FAA, 2011).

The newly proposed concept of operation is widely believed to change the fundamental authority structure of the current ATC system. Under some proposed concepts, pilots will bear the main responsibility of strategic conflict resolution while air traffic controllers are likely to act in a supervisory role, ensuring strategic flow management and resolving conflicts in exceptional situations where intervention is required on a tactical basis (Erzberger, 2004; Jha, Bisantz, & Parasuraman, 2003). However, some initial research in this direction has shown that air traffic controllers have difficulty in maintaining situation awareness under these conditions.

As the workload of the air traffic controllers becomes heavier and the tasks more complex, the need for a system that can support the air traffic controllers becomes greater (Banks, 2002). Extensive research has investigated the development of a conflict detection and resolution model that can identify conflicts and generate optimal resolution strategies that are conflict free. Researchers have emphasized that the CD&R models implemented in the DSTs will provide a functional reduction in air

traffic controller workload by decreasing the amount of time and mental effort needed to identify and resolve potential conflicts (Lanier & Coppenbarger, 2004). A large portion of the air traffic controller workload under the current system is identifying potential conflicts, generating a feasible resolution strategy, communicating with the pilot for strategy implementation, and monitoring the identified conflict pair to ensure commands are properly executed for resolution. Thus, if CD&R DSTs are to replace some of the mental workload of air traffic controllers, support tools will likely be required to accomplish the tasks identified.

Previous studies have tried to determine what level of automation would enhance the performance of air traffic controllers (Nijhuis, 2000). Different levels were considered as options, from fully manual to fully automated, with many levels in between (Sheridan & Parasuraman, 2005). The results from one of the studies showed that the best level of performance was achieved when automation provided the air traffic controllers with advisories and the controls necessary to decide whether to accept the advisory or not (Kirwan et al., 2001). An approach that seemed to work particularly well was when the resolution advisory of a potential conflict was derived from a mental model of the air traffic controller, as opposed to being derived from a purely mathematical model (Kirwan & Flynn, 2002). Many DSTs have been developed in an effort to effectively support ATCs in practice; however, the system cannot be prone to errors in case the human-machine relationship breaks down.

Different methods have been suggested for modeling the CD&R algorithm, mostly applying the tools of mathematical optimization or a probabilistic approach. Some of the early work for modeling CD&R algorithms has approached the problem by applying methods such as constrained optimization, differential equations, and Markovian processes (Bilimoria, Sridhar, & Chatterji, 1996; Blom, Bakker, Everdij, & Van der Park, 2003; Kosecka, Tomlin, Pappas, & Sastry, 1997; Rong, Geng, Valasek, & Ioerger, 2002). There are also probabilistic CD&R approaches that use stochastic methods to estimate the probability of conflict or collision (Campos & Marques, 2010; Kim, Lee, Lee, & Kang, 2013; Prandini, Hu, Lygeros, & Sastry, 2000; Prandini & Watkins,

2005; Vaddi, Kwan, Fong, & Cheng, 2012). In a comprehensive survey of the CD&R models, it was noted that the proposed CD&R algorithms are only applicable under a limited set of conditions and consider only limited options for resolving potential conflicts (Kuchar & Yang, 2000). Also, the efficiency and effectiveness of these algorithms remain to be established because practical implementation has not yet been achieved.

Given the likelihood that the air traffic controller will continue to play an important role in tactically managing air traffic in future operations, CD&R algorithms will need to be designed so that the air traffic controllers can be kept well integrated within the system. Developing a CD&R model based on air traffic controller knowledge can generate advisory resolutions that are much more acceptable to controllers. However, it is difficult for the system developer to acquire and understand the knowledge content of air traffic controllers' because the knowledge content is highly specialized. In attempts to acquire the basic knowledge of air traffic controllers, many previous researchers have investigated their working processes and practices. One of the research approaches was to model specific performances of air traffic controllers through simplified air traffic control simulation experiments. For example, researchers conducted experimental studies to analyze features of air traffic controllers' judgment under potential conflict (Loft, Bolland, Humphreys, & Neal, 2009; Rantanen & Wickens, 2012). Some work focused on how air traffic controllers dynamically adapt control strategies under different levels of workload (Fothergill & Neal, 2008). Another approach focused on the cognitive process of air traffic controllers and on developing a comprehensive description of their work (Endsley & Rodgers, 1994; Kallus, Van Damme, & Dittmann, 1999; Niessen, Eyferth, & Bierwagen, 1999; Rantanen, Yang, & Yin, 2006).

Although researchers have used different approaches to try to capture the mental model of air traffic controllers, research attempting to analyze their control strategies during real-world operation is rare (Fothergill & Neal, 2008). Previous studies reveal that, under controlled experimental conditions, experienced air traffic controllers show

performances that seem contradictory to what is expected. It has been reported that the conflict judgments do not seem to significantly differ between expert and novice air traffic controllers when dealing with traffic situations that are less complex (Rantanen & Nunes, 2005).

However, research has revealed how differential experiences affect the cognitive processes of air traffic controllers in managing traffic under varied situations (Redding, Cannon, & Seamster, 1992; Stankovic, Raufaste, & Averty, 2008). This finding could imply that the expertise of air traffic controllers is less noticeable when they are performing simple tasks and more evident in situations where sophisticated cognitive strategies are required. The air traffic controllers adapt their control strategies and the timing of input in relation to the level of the workload. It has been noted that a control strategy implemented by an air traffic controller can have an impact in the later phases of an air traffic situation (K. Inoue, Ando, Aoyama, & Yamato, 2006). Furthermore, adopting a control strategy that can predictively avoid conflict between aircraft was emphasized by ATC instructors as a way to reduce controller workload and increase traffic efficiency (Karikawa, Takahashi, & Aoyama, 2010). A field survey reported that air traffic controllers emphasized the importance of planning skills to manage the traffic situations appropriately with limited cognitive resources (D'Arcy & Rocco, 2001). The findings from previous research provide some valuable information about capturing the knowledge of air traffic controllers and addressing human factor issues in improving the aviation system. However, limited research is being conducted that examines control strategies and their effect on task demands (Fothergill & Neal, 2008). Therefore, research examining how air traffic controllers perform in a real operation environment is necessary to obtain a better understanding of the mental model they apply in actual practice. This kind of knowledge is relatively difficult to gather because air traffic controllers' control strategies for task demands are based on identification of future projections of the traffic situation and deal with analyzing control strategies that are difficult to interpret without explicit knowledge of air traffic control operations. Moreover, air traffic controllers are known to set

a buffer zone for potential error in their mental picture of future traffic situations because unexpected blunders can make it practically impossible to accurately project future traffic situations (Dwyer & Landry, 2009; S. J. Landry, 2011). The complex cognitive processes of air traffic controllers described above make it extremely difficult for researchers to analyze their dynamic task processes when operating in an actual operational environment.

Based on the information gathered, it is important that we develop a method that can be utilized to explore and better understand when and how air traffic controllers intervene with traffic under real-world operation. Also, a generalization of air traffic controller knowledge is necessary when forming a representation of air traffic controllers' mental model. This knowledge can then be used in the development of human-centered systems to better support the air traffic controllers. To date, no study has looked into the usage of historical data in an attempt to capture air traffic control strategies in dealing with traffic under operations. Recreating actual traffic situations of the past can be used to enable us to observe how air traffic controllers operate in actual practice in a practical manner.

3. METHODOLOGY

I undertook the following six procedural steps to develop the ATC intervention detection algorithm and analyze the aircraft deviations:

1. Filter FlightAware data to collect closed-loop trajectories with their original flight plan and generate corresponding open-loop trajectories based on the extracted flight plan information.

2. Identify aircraft pair(s) within the open-loop trajectory data that are expected to have an encounter in a loss-of-separation incident.

3. Compare and analyze the open- and closed -loop trajectories of those aircrafts that deviated due to expected loss of separation, and define operationally significant flight path deviation based on the analysis.

4. Classify aircraft(s) within the closed-loop trajectories as "deviated trajectory" and "non-deviated trajectory."

5. Calculate statistics for "deviated trajectory" and "non-deviated trajectory" categories to characterize flight path deviation due to ATC interventions for separation assurance.

6. Develop an algorithm to detect deviations in the flight trajectory data with respect to ATC interventions for separation.

3.1 Flight Trajectory Data

I identified a list of 162 aircraft operating within the Cleveland Air Route Traffic Control Center (ZOB ARTCC) at 1:00 p.m. on July 25, 2013. I collected the closed-

loop trajectory data of these aircraft through the FlightAware database, which is compiled from 45 government sources, multiple airlines, and commercial data providers, providing the flight profile with latitude, longitude, altitude, speed of each airborne flight in ZOB ARTCC, and a list of the navigation fixes that the flights intended to follow. I then utilized this information to generate open-loop trajectory data that corresponded to the list of closed-loop trajectory data collected from the FlightAware database. Figures 3.3.(a) and 3.3.(c) show example datasets that ware collected from the FlightAware database.

Figure 3.1. The cockpit view of the Microsoft Flight Simulator X (FSX).

I used Microsoft©Flight Simulator X (FSX) software for generating the open-loop trajectory data. I selected FSX for the simulation because it could be easily

customized utilizing open source software, and it also provided the necessary quantitative output for the trajectory data. Figures 3.1 and 3.2 show examples of visual displays that microsoft©Flight Simulator X (FSX) software provides.

To generate the open-loop trajectory data, I set zero wind and standard temperature to eliminate external factors that could possibly affect the output of the simulation.

The re-creation of real traffic situations using open-loop flight trajectory data approximated the actual air traffic as if aircraft in the airspace flew irrespective of each other and without controller input. It attempts to replicate potential conflicts that would be commonly resolved in real-world operations. This enables us to characterize the conflict-event process within an airspace and is especially useful for characterizing ATC intervention when resolution is needed for the expected events. I chose the ZOB ARTCC as the test environment because it has one of the busiest and most complex air traffic environments in the U.S. national airspace (Dale, 2013). Both closed-loop and open-loop trajectory data are interpolated to 60 second intervals.

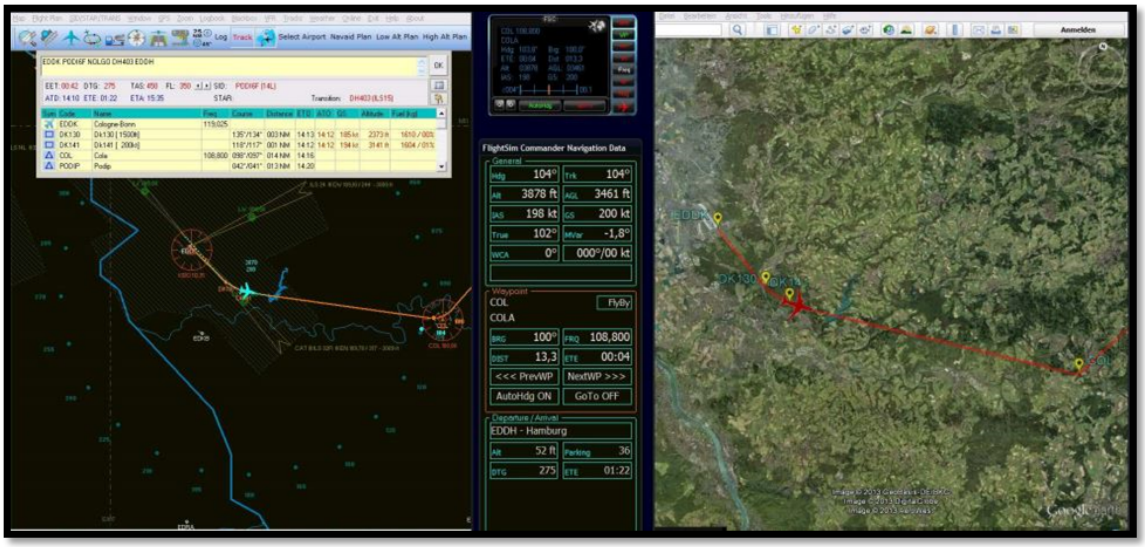
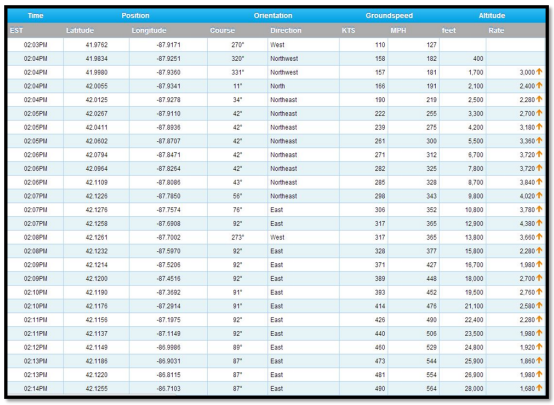
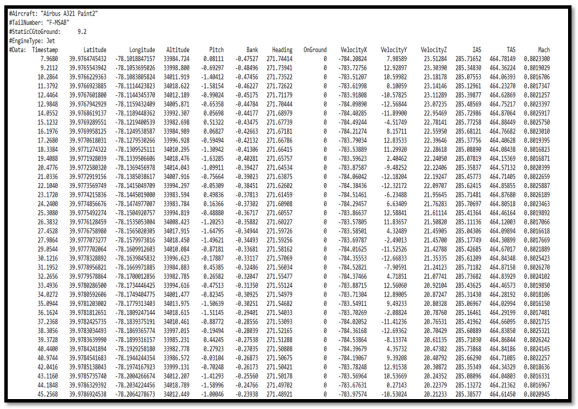


Figure 3.2. Display of the FSX generating open-loop trajectory data with flight plan input.



Time	Position		Orientation		Groundspeed		Altitude	
EST	Latitude	Longitude	Course	Direction	KTS	MPH	feet	Rate
02:03PM	41.9762	-87.9171	270°	West	110	127		
02:04PM	41.9834	-87.9251	320°	Northwest	158	182	400	
02:04PM	41.9980	-87.9360	331°	Northwest	157	181	1,700	3,000
02:04PM	42.0055	-87.9341	11°	North	166	191	2,100	2,400
02:04PM	42.0125	-87.9278	34°	Northeast	190	219	2,500	2,200
02:05PM	42.0267	-87.9110	42°	Northeast	222	255	3,300	2,700
02:05PM	42.0411	-87.8936	42°	Northeast	239	275	4,200	3,180
02:05PM	42.0602	-87.8707	42°	Northeast	261	300	5,500	3,360
02:06PM	42.0794	-87.8471	42°	Northeast	271	312	6,700	3,720
02:06PM	42.0964	-87.8264	42°	Northeast	282	325	7,800	3,720
02:06PM	42.1109	-87.8086	43°	Northeast	285	328	8,700	3,840
02:07PM	42.1226	-87.7850	56°	Northeast	298	343	9,800	4,020
02:07PM	42.1276	-87.7574	76°	East	306	352	10,800	3,780
02:07PM	42.1259	-87.6908	92°	East	317	365	12,900	4,380
02:08PM	42.1261	-87.7002	273°	West	317	365	13,800	3,660
02:08PM	42.1232	-87.5970	92°	East	328	377	15,800	2,280
02:09PM	42.1214	-87.5206	92°	East	371	427	16,700	1,980
02:09PM	42.1200	-87.4516	92°	East	389	448	18,000	2,700
02:10PM	42.1190	-87.3692	91°	East	393	452	19,500	2,760
02:10PM	42.1176	-87.2914	91°	East	414	476	21,100	2,580
02:11PM	42.1156	-87.1975	92°	East	426	490	22,400	2,280
02:11PM	42.1137	-87.1149	92°	East	440	506	23,500	1,980
02:12PM	42.1149	-86.9986	89°	East	460	529	24,800	1,920
02:13PM	42.1186	-86.9031	87°	East	473	544	25,900	1,860
02:13PM	42.1220	-86.8115	87°	East	481	554	26,900	1,980
02:14PM	42.1255	-86.7103	87°	East	490	564	28,000	1,680

(a) Example of closed-loop trajectory data.



(b) Example of open-loop trajectory data (FSX).

KORD DUFEE ELX HAAKK DOXXY SOSIC JHW J70 LVZ LENDY6 KJFK					
Name	Latitude	Longitude	Distance from origin	Distance from destination	Type
KORD	41.9793333	-87.9073889	0	739	Origin Airport
DUFEE	42.1124722	-87.0633333	44	696	Waypoint (RNAV)
ELX	42.1444444	-86.1227778	92	648	VOR-DME (NAVAID)
HAAKK	42.1333333	-85.0333333	148	593	Reporting Point
DOXXY	42.1225000	-84.4736111	177	564	Waypoint (RNAV)
SOSIC	42.2297528	-83.4908278	227	515	Reporting Point
JHW	42.1886075	-79.1213050	451	297	VOR-DME (NAVAID)
HOXIE	41.8650111	-77.8526056	517	228	Reporting Point
DMACK	41.7857583	-77.5519222	533	212	Reporting Point
STENT	41.6790472	-77.1527556	554	190	Reporting Point
MAGIO	41.5273833	-76.5964944	584	159	Reporting Point
LVZ	41.2728025	-75.6894678	633	109	VOR-TAC (NAVAID)
LVZ	41.2728056	-75.6894722	633	109	VOR-TAC (NAVAID)
JENNO	41.1529167	-75.3314167	653	89	Reporting Point
HARTY	41.0711944	-75.0898889	666	75	Reporting Point
MUGZY	41.0301389	-74.9693611	673	68	Reporting Point
STW	40.9958333	-74.8690278	678	62	VOR-DME (NAVAID)
LENDY	40.9148333	-74.1352500	717	27	Reporting Point
LGA	40.7837222	-73.8686111	732	11	VOR-DME (NAVAID)
KJFK	40.6397511	-73.7789256	739	0	Destination Airport

(c) Example of Waypoint Information (FlightAware).

Figure 3.3. Example Dataset for Analysis.

3.2 Conflict Detection

An aircraft conflict or loss of separation occurs when minimum separation is not maintained between two or more aircraft at all times. The specific separation requirements vary depending on the phase of flight that the aircraft is operating under and the type of operation that the aircraft is involved in, but typically, the Federal Aviation Administration (FAA) specifies that any two aircraft must maintain at least 5 nautical miles (NM) horizontally and 1,000 feet vertically during the en-route phase of operation, as illustrated in Figure 3.4. (Zhao & Schultz, 1997).

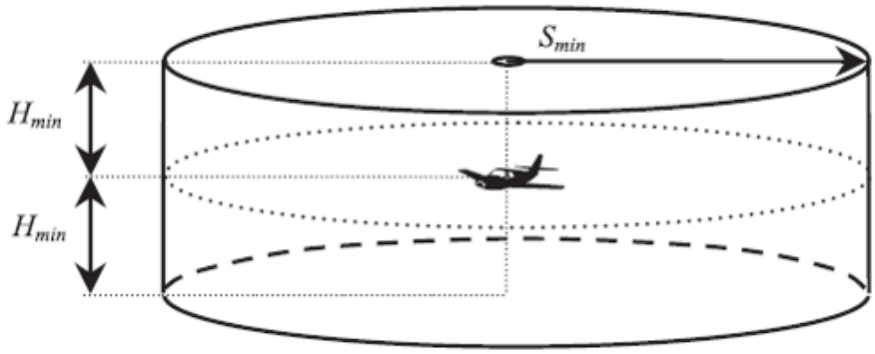


Figure 3.4. Illustration of the Aircraft Protection Zone. Adapted from (Netjasov, 2012).

In practice, all aircraft operating within controlled airspace in the U.S. National Airspace System are constantly monitored by air traffic controllers. In order to ensure that the minimum separation standards are maintained by all aircrafts under operation, their flight trajectories are anticipated by the controller and potential conflicts are detected. When a potential conflict is identified, the controllers will communicate with the aircraft to initiate appropriate conflict avoidance maneuvers.

The air traffic controller directs different types of control actions to one or more aircraft that are encountering a potential conflict to prevent the expected event from taking place. The types of control action include deviation applied to an aircraft's horizontal path, vertical path, and in speed (Rantanen & Wickens, 2012). Each type

of control action in resolving a conflict has its advantages and limitations. When deciding which control action to apply, the controller must take into account factors such as the time remaining until the conflict, aircraft performance capabilities, and traffic situations in the airspace. When necessary, multiple control actions can be applied to avoid a conflict.

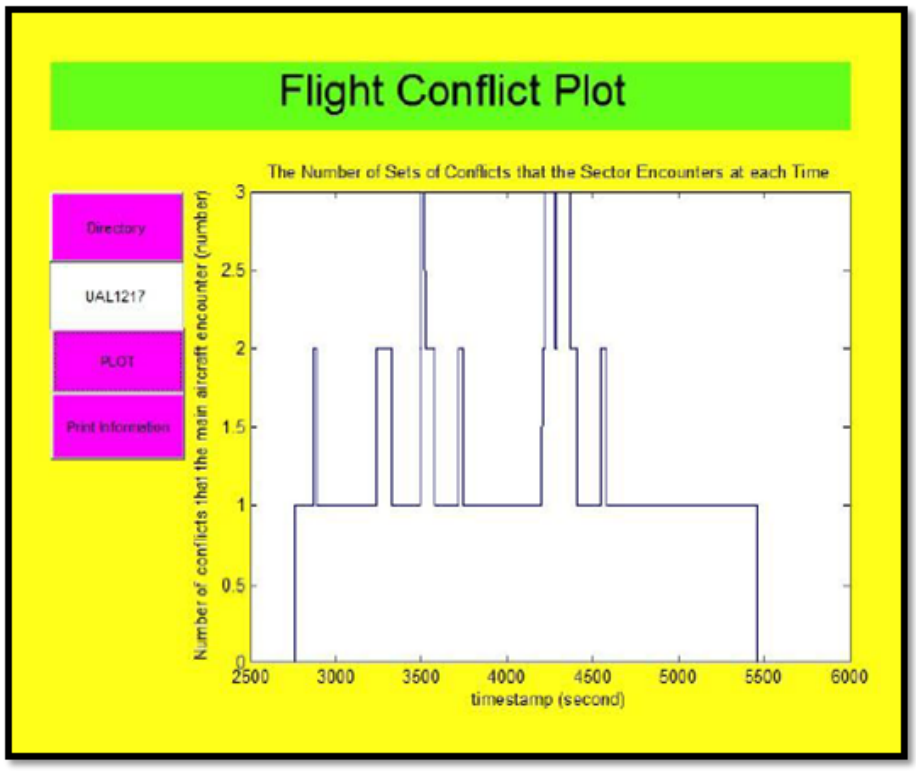


Figure 3.5. A flight conflict plot that shows the number of conflict encounters for aircraft "UAL1217."

For the purposes of determining air traffic controller interventions associated with the conflict resolution process, controllers check relative aircraft distances to determine whether aircraft pairs come in close proximity. If a violation of minimum separation standards is identified, control action by the controller is expected to take place. I identify the aircraft pairs that are expected to lose separation by implementing an algorithm that calculates the aircraft separation distance by using a Matlab program. The algorithm also identifies the total number of conflicts that a single

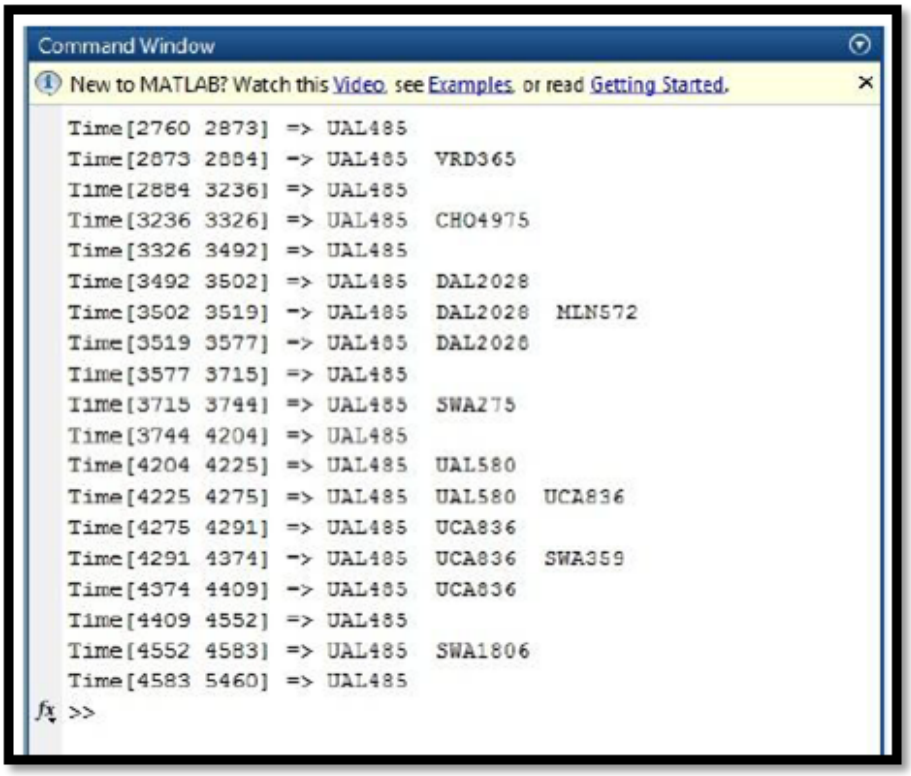


Figure 3.6. List of aircrafts in a conflict encounter with "UAL1217."

aircraft encounters for the time it enters and exits an airspace sector and the time duration of the expected conflict. The aircraft trajectory of A_x is denoted ($x_i(t)$, $y_i(t)$, $z_i(t)$), representing the position and altitude of the aircraft over time. The conflict is identified for two aircrafts A_i and A_j when the relative horizontal and vertical separation distances satisfy both conditions of $D_{i,j}(t)$, at any point in time.

$$D_{i,j}(t) = \begin{cases} r = \sqrt{\left(x_i\left(t\right) - x_j\left(t\right)\right)^2 + \left(y_i\left(t\right) - y_j\left(t\right)\right)^2} \ < 5NM \\ h = \left|z_i\left(t\right) - z_j\left(t\right)\right| \ < 1000FT \end{cases} \tag{3.1}$$

Any conflict that was detected to occur below 18,000 feet was not considered in the analysis to ensure that the aircrafts were in the en-route portion of the flight phase, where nonstandard maneuvering is at a minimum. Figures 3.5 and 3.6 show examples of how the Matlab algorithm displays the results for an aircraft that was identified to have an encounter in a loss of separation incident.

3.3 Deviation Categorization

The open-loop trajectories of the aircraft that the algorithm identified would encounter a conflict were compared to the corresponding closed-loop trajectories. After completing a thorough analysis of each aircraft pair in conflict, I classified each aircraft within the conflict pair as "deviated trajectory" or "non-deviated trajectory." Figures 3.7 and 3.8 present illustrations of how the trajectories were categorized.

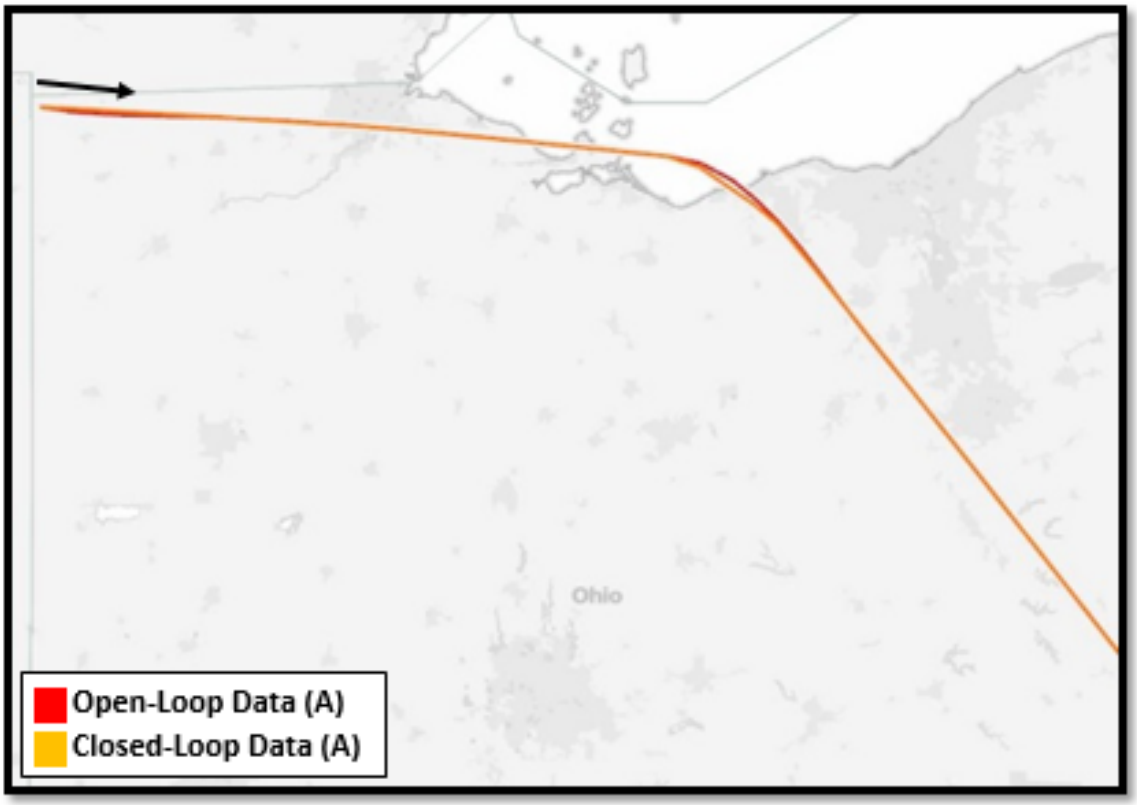


Figure 3.7. Example of a "non-deviated trajectory."

Trajectories were classified as "deviated trajectory" only if the horizontal flight path deviation appeared to resolve the expected loss of separation. Figure 3.9 illustrates an example of an operationally significant horizontal deviation.

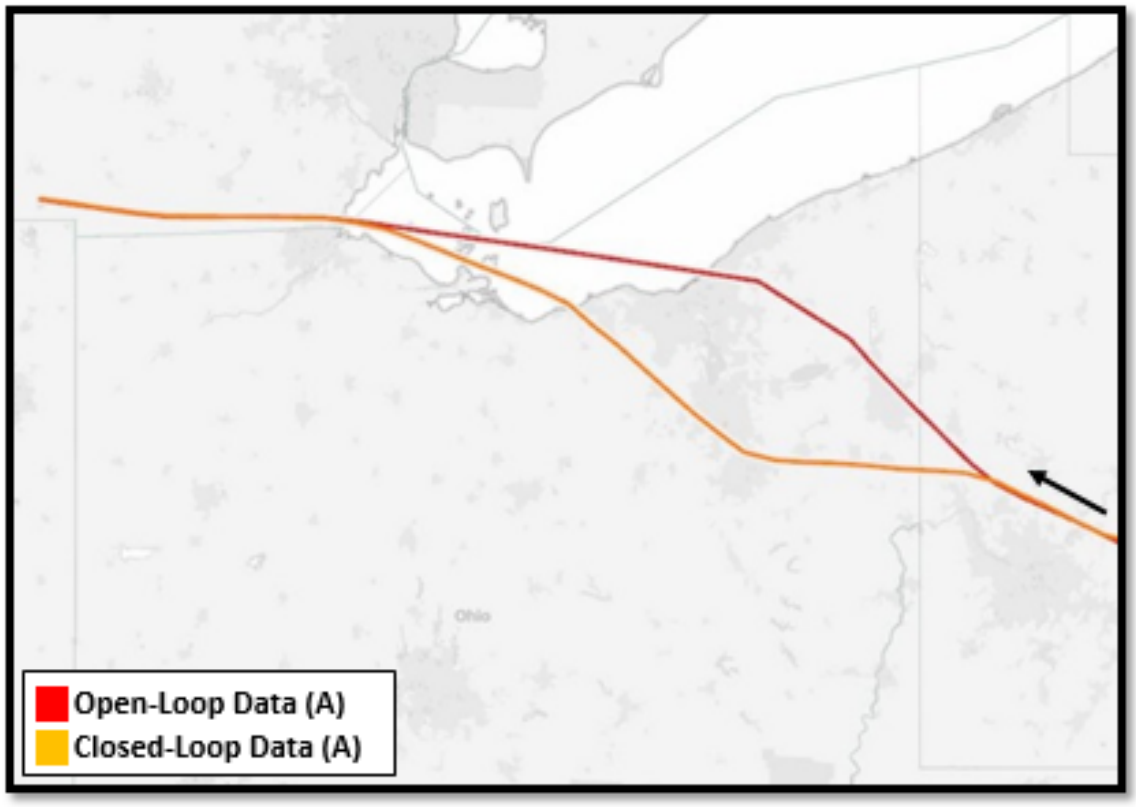


Figure 3.8. Example of a "deviated trajectory."

I also examined and categorized the remaining data that were not identified as being in potential conflict. If deviation was noticed among trajectories that were not in the conflict set, I pulled those trajectories out of the analysis to ensure that any deviations occurred solely due to ATC intervention and not because of other intentions that were unknown in the experiment. Figure 3.10 shows an example of a deviation that occurred for unknown reasons.

The horizontal deviation distance of the aircraft from its filed flight path was calculated for the categorized trajectories and only the maximum deviation distance values were collected as data for the "deviated trajectory" category. The statistical measures from the categorized data provided a deviation threshold, which represents the limits of normal operational variation in flight trajectories along its planned flight

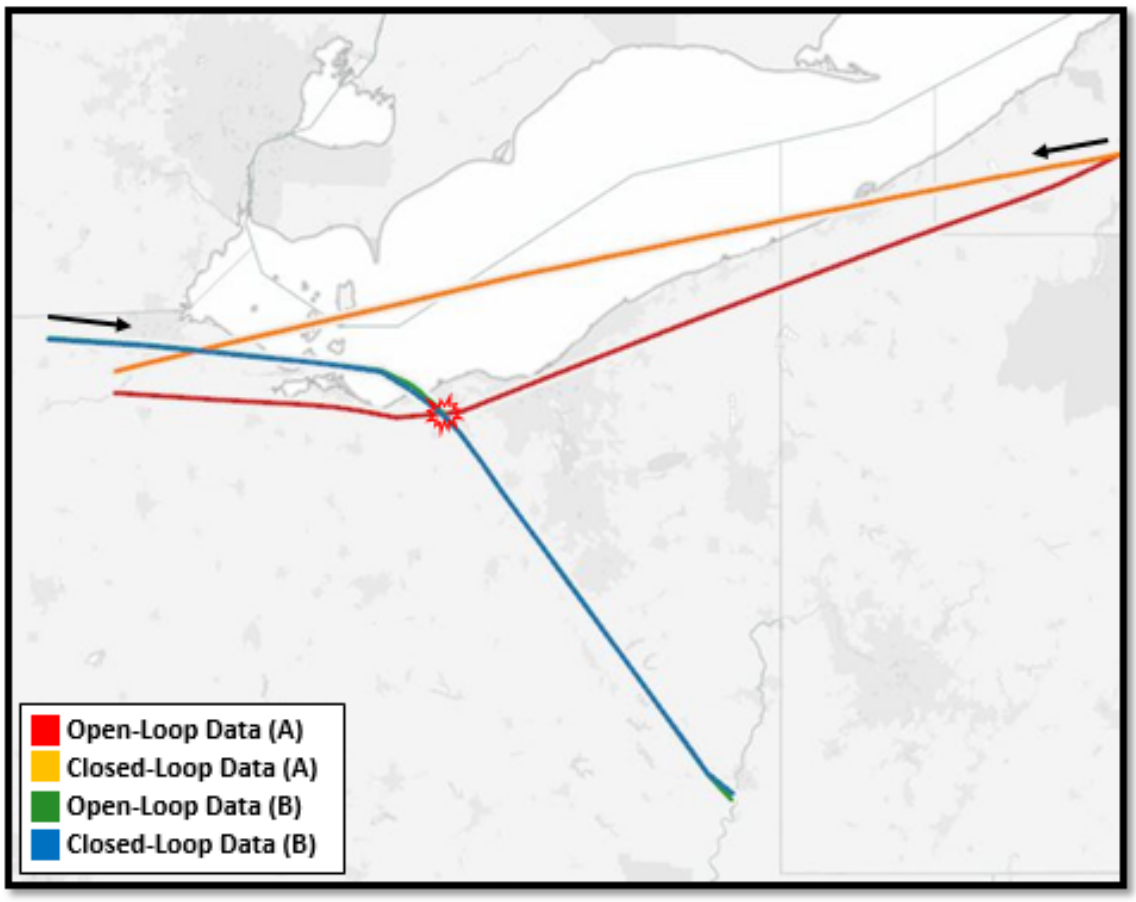


Figure 3.9. Example of an operationally significant horizontal deviation.

path, for the deviation detection model. The details regarding the statistical measures that provided the deviation threshold will be discussed in section 4.2.

3.4 Horizontal Deviation Measurement Method

Deviation is defined as a flight trajectory in which the predetermined deviation distance is greater than some deviation threshold. The deviation distance is the distance from each point on the open-loop trajectory to the nearest point on the closed-loop trajectory. But, if the nearest points for the two trajectory data sets are

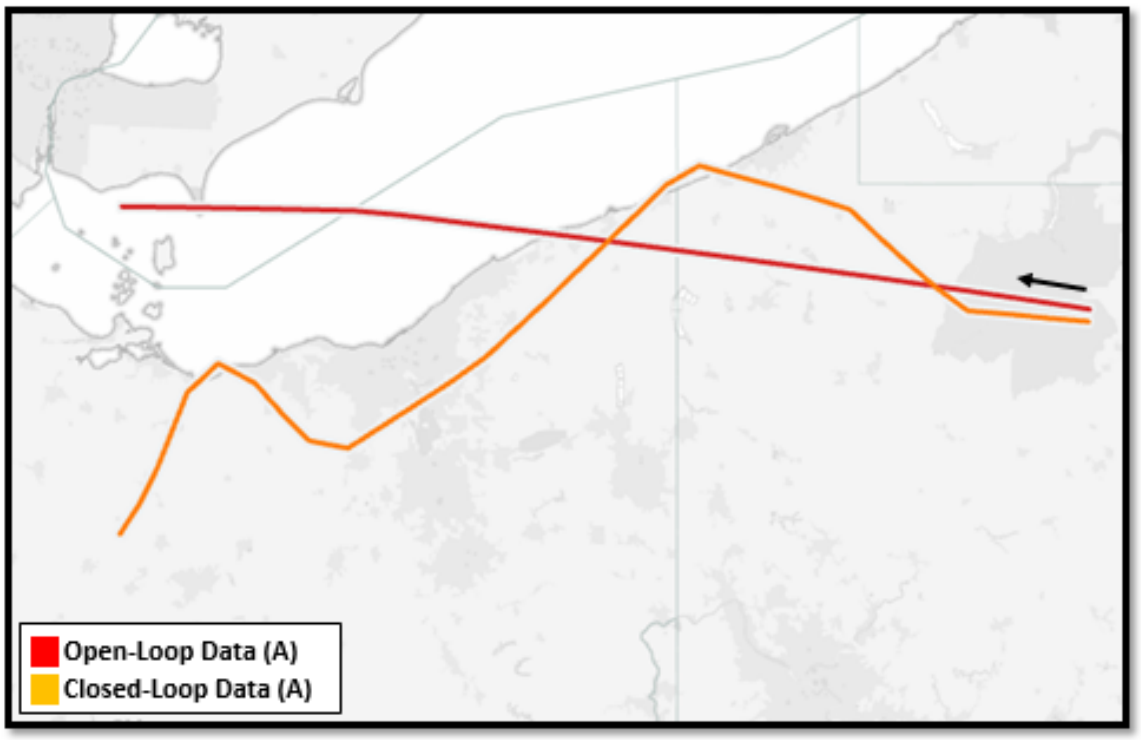


Figure 3.10. Example of a deviation with unknown intention.

not perpendicularly aligned, the measurement can be sensitive to along-track error. Therefore, in this study, I also measured the dot product of the two nearest vectors to see whether the two different measuring methods had any significant difference in providing a good measure for the deviation threshold.

3.4.1 Method 1.

One way to measure horizontal deviation of an aircraft in regards to its filed plan is by measuring the aircraft's relative distance from its planned flight path. The angle of two vectors from some point of origin can act as a good measure in representing relative distance. The spherical coordinates determine the position of a point that is constructed in a three-dimensional euclidean space based on the distance from the

orgin and two angles ϕ and θ, as illustrated in Figure 3.11. The two points on a sphere can be considered as vectors defined by latitude, longitude, and radius. The Cartesian coordinates for the defined vectors are x,y,z in the standard right-hand coordinate system.

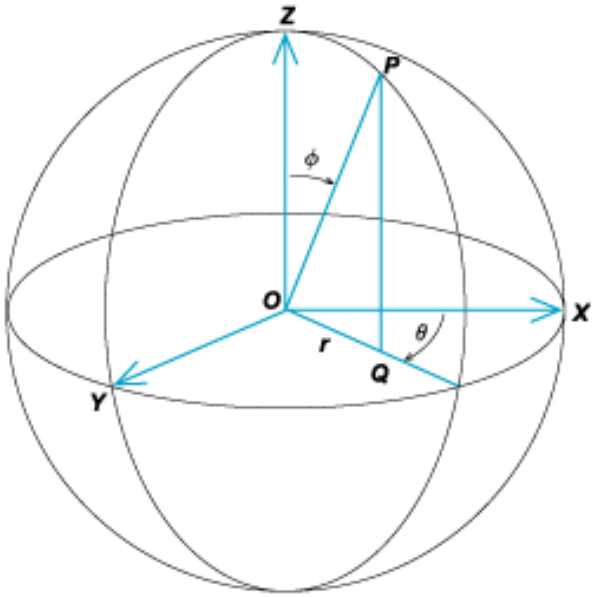


Figure 3.11. Graphic depicting the spherical coordinate system. Adapted from (McGraw-Hill, 2009).

The origin is considered to be the center of the earth, and it is assumed that the earth is spherical. From the spherical coordinate figure, the x, y, and z components of the Cartesian coordinate system can be calculated (Julier & Uhlmann, 1997).

$$The\ Cartesian\ coordinate\ (x, y, z) = (rcos\varphi sin\theta, rsin\varphi sin\theta, rcos\theta) \tag{3.2}$$

The dot product can be applied to calculate the cos(δ), where (δ) is the angle between the two vectors.

$$\cos(\delta) = \frac{A_x B_x + A_y B_y + A_z B_z}{|\overrightarrow{A}||\overrightarrow{B}|} \tag{3.3}$$

The cos(δ) value will indicate whether the aircraft is conforming to the planned flight path or deviating away from it by providing a value between -1 and 1 (Krozel, 2002).

3.4.2 Method 2.

There is no method that can calculate the exact distance between two geographical coordinates due to the irregularity in the surface of the earth. There are several methods that can be used for the purposes of this study. The spherical law of cosines, Haversine, and Vincenty formulas are a few that are widely used to calculate the distance between coordinates. Whereas the Vincenty formula is used to calculate the ellipsoidal distance between two points on the surface of a spheroid, the spherical law of cosines and Haversine formulas calculate the great-circle distance, assuming that the earth is spherical (Pineda-Krch, 2010). I used the spherical law of cosines in this study because the results from the calculation were identical to the results obtained from the Haversine formula, and its computational implementation was simpler. The formula provides well-conditioned results as long as the distance between the two coordinates is not smaller than 1 meter (Williams, 1997). The following is the spherical law of cosines formula that I used to calculate the distance from each point on the open-loop trajectory to the nearest point on the closed-loop trajectory (Junkins & Shuster, 1993):

$$d = \text{acos}(\sin(\varphi_1) \ * \sin(\varphi_2) \ + \cos(\varphi_1) * \cos(\varphi_2) \ * \cos(\Delta\lambda) \ * R \qquad (3.4)$$

where ϕ is latitude, λ is longitude, and R is the earth's radius. It was assumed that the mean radius of the earth is 3440.065 NM. The deviation measurement of aircraft A and aircraft B is shown in Figure 3.12 using both of the methods mentioned in this section. The two methods present identical patterns in detecting deviation, but measuring the angle between two coordinates results in a smoother graph.

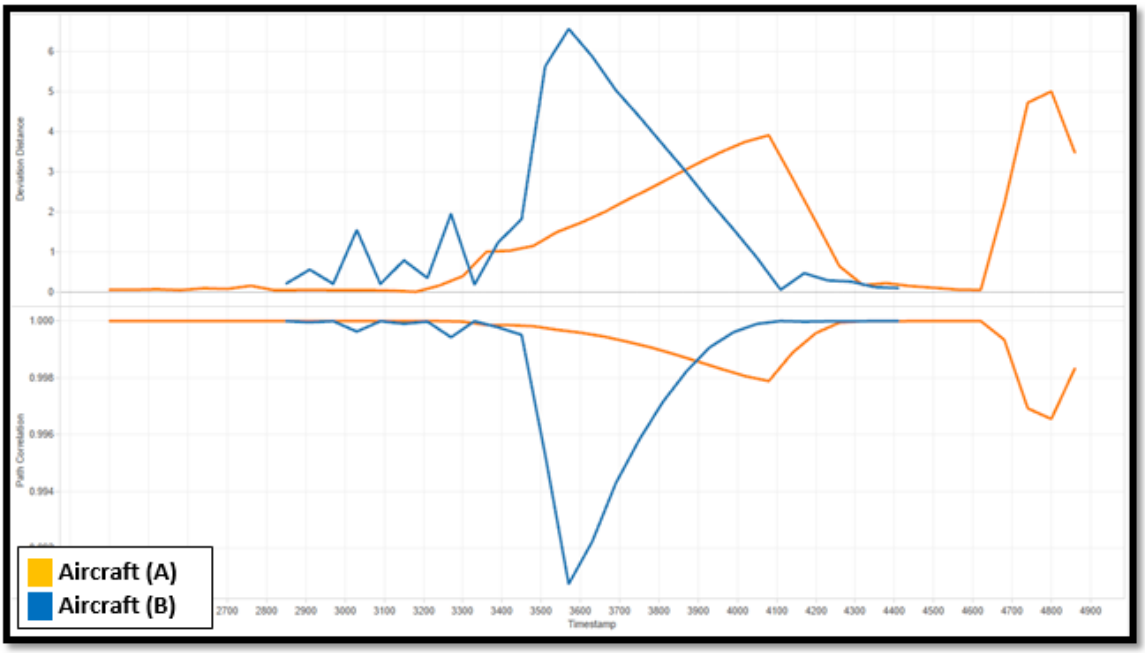


Figure 3.12. Example of deviation measurement in distance and angle.

3.5 ATC Intervention Detection Algorithm

The proposed heuristic algorithm that can detect ATC intervention due to predicted conflict is provided in Figure 3.13. It begins by calculating the deviation distance (or angle) of the aircraft from its filed flight path, and the local maximum of the deviation value identified in the deviation segment is denoted D^*. The local maximum deviation value is then compared to the threshold value for the aircraft in conformance to the planned flight path, and if it is smaller than the threshold D_n, then it is classified as a "non-deviated trajectory." If the local maximum deviation value is larger than the threshold D_n, then it is checked against a larger threshold D_d. If it exceeds this threshold, it is classified as "deviated trajectory" due to ATC intervention in order to prevent predicted conflict from taking place.

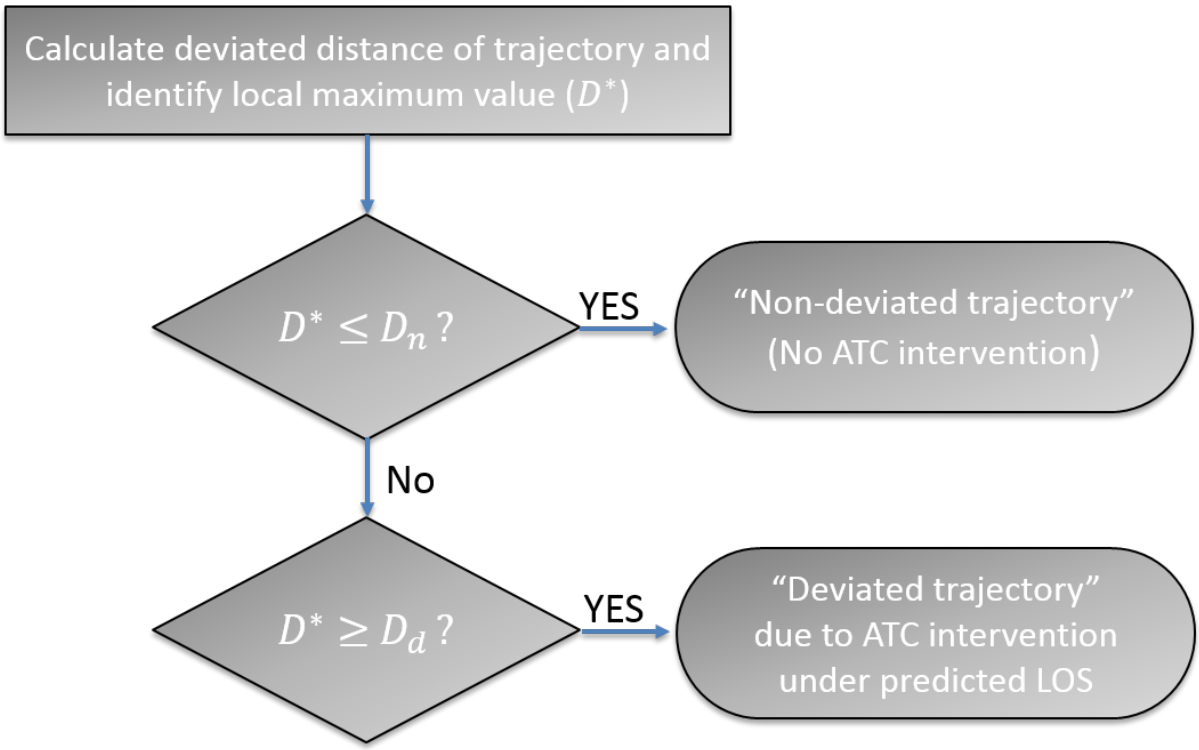


Figure 3.13. Heuristic states of horizontal deviation.

4. RESULTS AND ANALYSIS

4.1 Conflict Detection Algorithm

From the 162 open-loop trajectory data inputs, I identified a total of 47 pairs of aircraft that would encounter a loss of separation if no control action were taken by the air traffic controller.

Table 4.1
Partial list of aircraft pairs that were identified to encounter conflict

AC 1	AC 2	Lat 1	Long 1	Lat 2	Long 2	Timestamp	
ASH3734	SWA3074	41.28635	-80.8123	41.23886	-80.8778	5075	5085
FFT532	SWA1806	41.54583	-81.4746	41.58517	-81.3668	5040	5079
EGF3040	SWA359	41.4197	-82.2479	41.36155	-82.157	4872	4051
EGF3040	LOF3395	41.53861	-82.6526	41.56379	-82.5419	4700	4736
SWA359	UAL358	41.51822	-81.6039	41.57781	-81.6927	4658	4686
UAL1217	SWA1806	41.75882	-80.1594	41.9324	-80.1005	4552	4583
LOF3409	TCF3517	42.4857	-80.3407	42.56939	-80.2965	4360	5021
UAL1217	SWA359	41.83042	-80.8815	41.78303	-80.652	4292	4374
SWA359	UAL485	41.78723	-80.6365	41.81562	-80.7278	4286	4318
SWA275	UAL1037	42.00458	-83.9391	41.94255	-84.0097	4259	4288
UAL1217	UAL580	41.85355	-81.1255	41.92792	-80.9068	4204	4274
FLG3326	TCF7324	41.97295	-79.9481	42.00231	-79.837	4022	4418
NKS851	UPS9311	40.01552	-80.8127	40.01662	-80.6968	4017	4058
UAL580	UAL1105	41.80318	-80.0813	41.76014	-80.1706	3909	3939
SWA275	UAL1577	42.07643	-82.8733	42.07027	-82.9721	3878	3916
MLN572	SWA423	42.15766	-83.2756	42.19421	-83.3861	3758	3795
UAL1217	SWA275	41.92736	-82.4916	42.09685	-82.4122	3715	3744
MLN572	ASQ5278	42.10421	-83.1997	42.04268	-83.1968	3708	3720
CHQ4981	TCF3517	42.25696	-78.5841	42.28906	-78.472	3667	3705
MLN572	CHQ6379	42.03	-83.0948	42.07056	-83.1002	3641	3669
ASQ4444	CHQ4981	42.31611	-78.5781	42.24795	-78.6569	3639	3665

Table 4.2
List of aircrafts encountering multiple conflicts

Main	AC 1	AC 2	AC 3	AC 4	AC 5	AC 6
UAL580	UAL1105	UAL1111	UAL1217			
UAL566	AWE1839	DAL2428				
UAL1217	SWA275	SWA359	SWA1806	UAL485	UAL580	
TCF3517	CHQ4981	LOF3409				
SWA359	UAL358	UAL485	UAL1217	EGF3040		
SWA2789	EJA313	UAL1111				
SWA275	UAL1037	UAL1038	UAL1577	UAL1217	DAL1532	CHQ4717
SWA2149	AAL3	AAL1855	AWE1839	UAL703		
SWA1806	FFT532	UAL1217				
FLG3326	TCF7324	CHQ6378				
EJA313	SWA2789	AWE1839				
EGF3040	LOF3395	SWA359				
DAL2428	SWA2696	UAL566				
DAL1532	CHQ6104	SWA275				
CHQ4717	SWA275	UAL255				
AWE1839	AAL712	EGF2758	EJA313	SWA2149	UAL566	UAL1733

Among those pairs, 16 aircraft would have encountered multiple conflicts. The list of the aircraft pairs that were identified to encounter conflict(s) is presented in Tables 4.1 and 4.2. I analyzed the pairs identified in the algorithm for categorization.

4.2 Threshold for the ATC intervention detection algorithm

Next, the 162 en-route trajectories in ZOB were analyzed to determine the deviation threshold for the ATC intervention detection algorithm. From the methodology discussed earlier, the closed-loop data was classified into two categories: "non-deviated trajectory" and "deviated trajectory." I set the threshold for D_n and D_d using the 95th and 5th percentile value from the distribution of the two categories classified. I used the 95th and 5th percentile value assuming that the aircraft oper-

Table 4.3
Thresholds for Horizontal Path Conformance States

Threshold	**Value**	
	Distance	Angle [Cos(δ)]
$\mathbf{D_n}$	2.5189 NM	0.9988
$\mathbf{D_d}$	9.007 NM	0.9888

ating within the airspace will achieve navigation performance accuracy at least 95% of the time. The threshold values used in this study are listed in Table 4.3.

4.3 Validation of ATC intervention detection algorithm

The algorithm identifies deviation in the closed-loop flight trajectories that occurred due to air traffic controller intervention in resolving expected loss of separation, using the methods and thresholds described earlier. In order to verify whether or not the ATC had intervened with the pairs of aircraft identified in the Matlab algorithm, I analyzed the pairs to identify such deviations as a validation step.

Table 4.4
Deviation Detection and Event Combination

	ATC Intervention Occured	**ATC Intervention did not Occur**
Detected	Algorithm detects intervention and it occured. (valid detection)	Algorithm detects intervention and it did not occur. (false detection)
Not Detected	Algorithm does not detect intervention and it occurred. (missed detection)	Algorithm does not detect intervention and it did not occur. (correct no-calls)

A new list of 127 aircraft operating within the ZOB ARTCC at 12:00 p.m. on July 31, 2013, was compiled as a sample data set to validate the accuracy of the algorithm. The four situations that are probable in the algorithm are shown in Table 4.2.

Table 4.5
Performance Summary of Deviation Detection Algorithm

	Distance	**Angle**
False Detection	44	45
Missed Detection	3	3
Valid Detection	21	21
Verified Intervention Due to Conflict	24	
Verified Detection	65	66
Rate of Missed Detection	0.125	0.125
Rate of False Detection	0.677	0.682

$$* \text{ Rate of Missed Detection} = \frac{\text{Missed Detections}}{\text{Total Number of Detections (Valid Detections + False Detections)}}$$

$$* \text{ Rate of False Detection } = \frac{\text{False Detections}}{\text{Total Number of Detections (Valid Alerts + False Alerts)}}$$

From the 127 closed-loop trajectory data newly collected, the ATC intervention detection algorithm identified and classified 65 (66 using angle measurement) flight trajectories being deviated by the air traffic controller due to expected loss of separation. The Matlab conflict detection algorithm was able to verify through the open-loop data that there were 24 pairs of aircraft that actually had an encounter in a loss

of separation. The performance of the deviation detection algorithm is summarized in Table 4.3.

The results show that 21 valid detections were made by the algorithm with 3 detections missed.

The measurement method used in the ATC intervention detection algorithm does not seem to have a noticeable effect on the results. The algorithm achieved an 85.7% detection rate in detecting deviations made by the ATC for resolving identified conflicts, and the probability of the algorithm giving a false alarm was 68%, although the false alarm includes actual deviations that occured for reasons other than predicted conflict avoidance. However, analyzing the 24 identified conflicts revealed that 3 conflict cases that had not been detected were resolved by a vertical maneuver applied by the air traffic controller. All of the identified conflicts that had been resolved by a horizontal maneuver have been detected. The rate of false alarms was higher than anticipated. This could be for many unknown reasons, but one possibility comes from the fact that air traffic controllers are known to keep aircraft sufficiently separated so that there is enough time to intervene to prevent loss of separation when an unexpected event takes place (S. Landry & Kim, 2010).

4.4 Effect of increased horizontal spacing on algorithm performance

To examine whether the buffer zone in the air traffic controller mental model had any effect on the false alert rate, I examined the performance of the algorithm under a protection zone with increased horizontal spacing. Table 4.4 illustrates the effect the increased horizontal spacing of the aircraft protection zone had on the performance of the algorithm. I modified the Matlab algorithm constraint to detect aircraft conflicts under different conditions. The results reveal that the false alarm rate decreases as the protection zone becomes wider in horizontal range. The false alarm rate decreased to 0.292, while the missed detection rate maintained a similar level of accuracy when the horizontal radius of the protection zone was set at 8 NM, suggesting that air traffic

controllers separated traffic with the consideration of unexpected events occurring in mind.

Table 4.6
Algorithm Performance with Increased Horizontal Range of Protection Zone

	Horizontal Range of Protection Zone			
	5 NM	6 NM	7 NM	8 NM
False Detection	44	39	27	19
Missed Detection	3	3	4	5
Valid Detection	21	26	38	46
Verified Intervention Due to Conflict	24	29	42	51
Verified Detection	65			
Rate of Missed Detection	0.125	0.103	0.095	0.098
Rate of False Detection	0.677	0.600	0.415	0.292

4.5 Analysis of ATC conflict resolution strategies

There were a total of 162 aircraft data sets that were subject to the categorization process. The algorithm identified 47 pairs of aircraft as having an encounter in a loss of separation if no control action were to be taken by the air traffic controller. Most conflicts were resolved by a control action implemented to a single aircraft, but there were cases in which multiple aircrafts were involved in the conflict resolution strategy.

Figure 4.3 illustrates a case in which an air traffic controller implemented resolution strategies to both aircraft identified to have an encounter in an expected conflict. Figure 4.4 is a plotted graph of the horizontal path deviation for Figure 4.3.

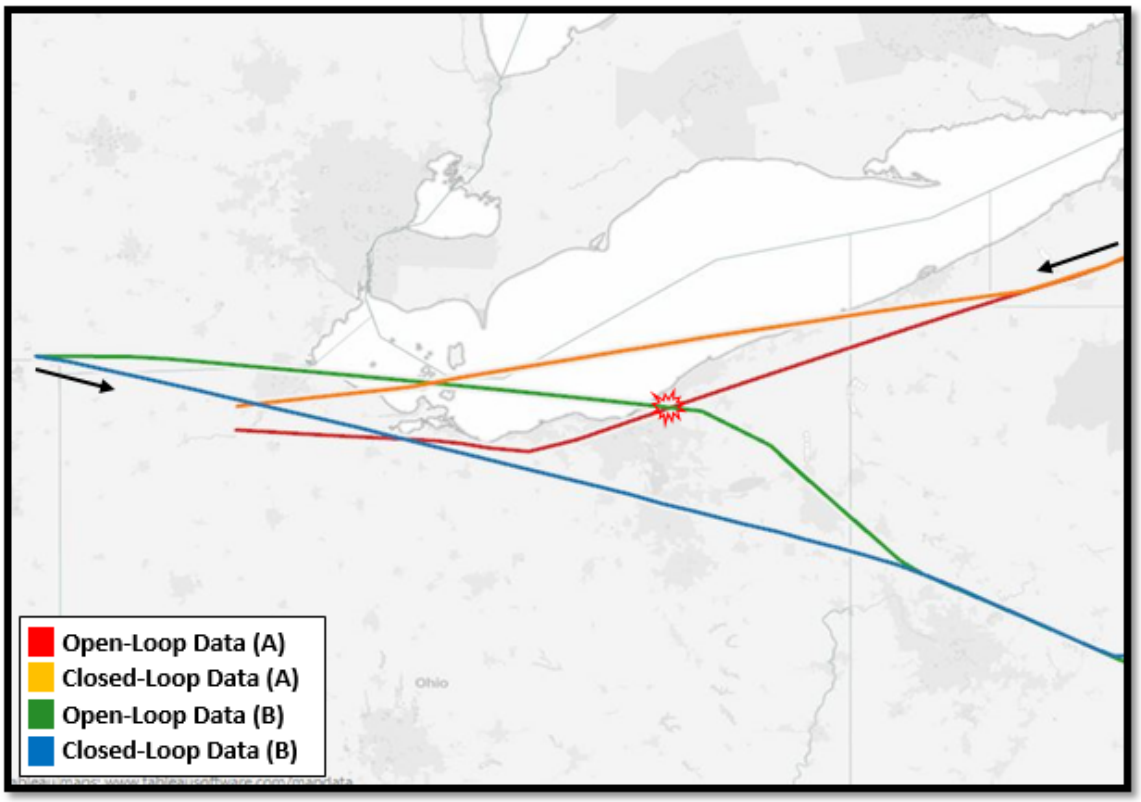


Figure 4.1. Example of a dual maneuver implemented in conflict resolution.

Among the 47 pairs of aircraft detected, 49 aircraft were strategically separated from the conflict by air traffic controllers implementing horizontal maneuvers. There were 2 cases in which a vertical maneuver was applied in resolving a potential conflict. Figure 4.5 provides an example of where an expected conflict between a pair of aircraft was resolved by a vertical maneuver. There is no evident horizontal maneuver applied to any of the aircraft in Figure 4.5, but Figure 4.6 clearly reveals that aircraft "B" deviated vertically to avoid loss of separation. Also, the small deviation (1 NM) that is apparent in Figure 4.6 was detected because aircraft flew past a waypoint

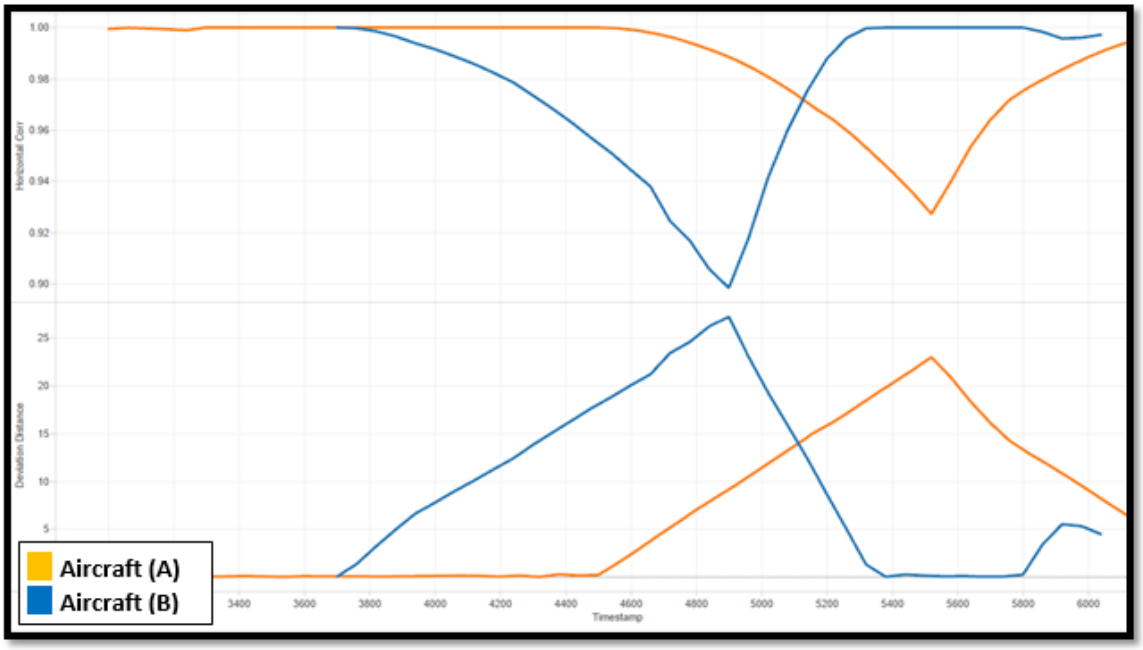


Figure 4.2. Deviation measurement of horizontal path for Figure 4.3.

in executing a necessary turn. The implementation of multiple types of maneuvers (horizontal + vertical) was identified in resolving the expected conflict. The cases in which a change in speed was the sole factor in resolving the conflict were not identified.

The ATC intervention detection algorithm in this study does not consider vertical deviations and speed changes in the detection because there were not enough sample cases in the resolution strategies to interpret for implementation of the two parameters. Also, the large amount of noise in the speed data made it difficult to identify a pattern from the information available. The study also revealed that 19 aircraft out of the 47 pairs were expected to have an encounter in more than one case of loss of separation. Figure 4.7 illustrates how "AWE1839" would have had an encounter in six cases of sequential conflict if the air traffic controller had not intervened.

Although "AWE1839" was not the only aircraft to have a control action implemented in resolving the sequential conflicts, it provided insight into the reason for

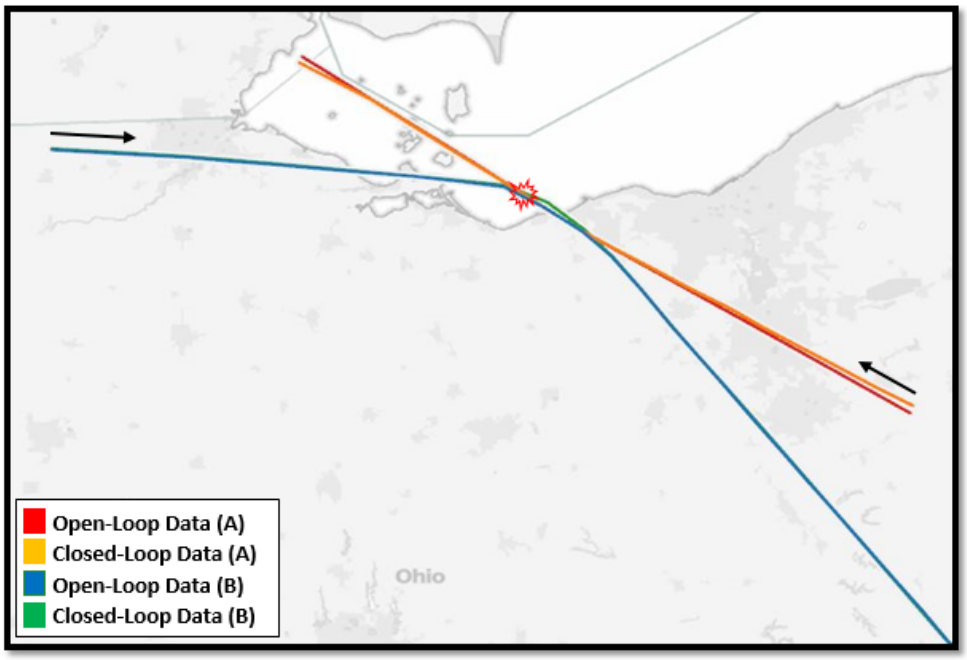


Figure 4.3. Example of a vertical strategy implemented in conflict resolution.

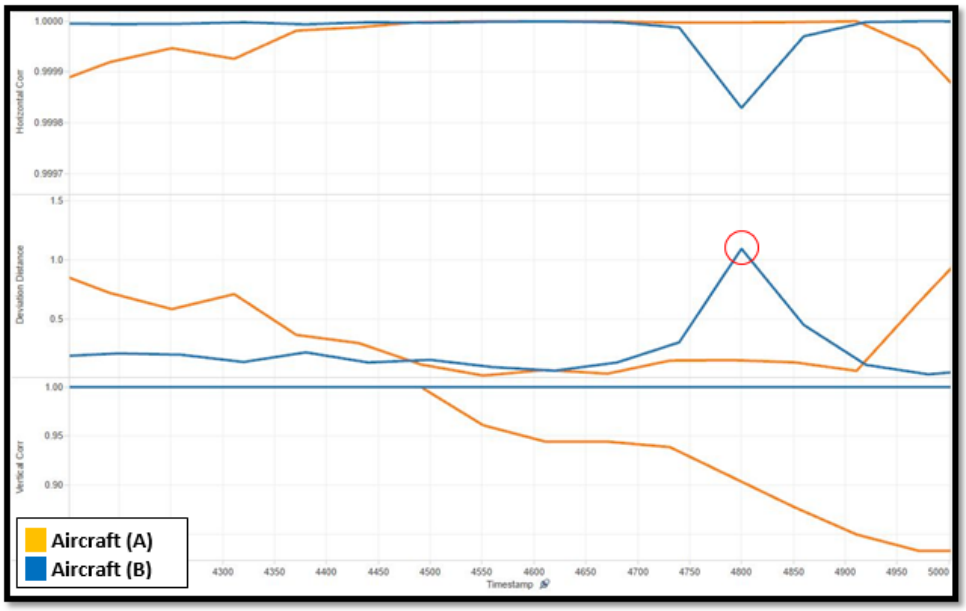


Figure 4.4. Deviation measurement of horizontal and vertical path for Figure 4.5.

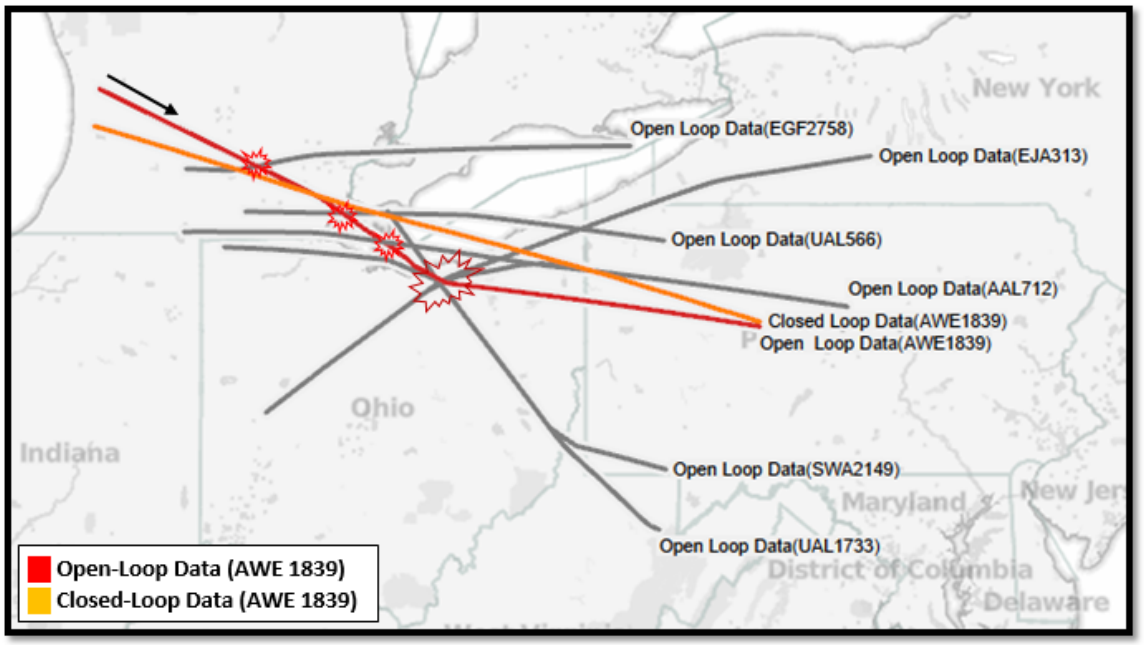


Figure 4.5. Illustration of aircrafts having encounters in multiple conflicts.

the large amount of horizontal deviation (26 NM) directed to "AWE1839" by the air traffic controller. The amount of deviation and the time the resolution strategy was initiated in the separation maneuver could be somewhat related to the number of conflicts that the aircraft was identified to encounter in the near future. Also, there was not a single case in which the horizontal maneuver resulted in an increased flight path. The air traffic controllers appear to facilitate the aircraft with maneuvers that enable greater economical accommodations for pilots. The time that the resolution maneuver was initiated varied from 5 to 26 minutes prior to the identified conflict, but the majority of the resolution strategies were initiated around fourteen to fifteen minutes prior to the conflict. The findings were somewhat similar to those in the previous study conducted by (Kirwan & Flynn, 2002), who identified air traffic controller conflict resolution strategies for CORA by conducting interviews.

5. CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

Human factors issues must be fully addressed to ensure that safety and efficiency are not compromised by implementing new technology to the airspace system. Although human factors researchers have emphasized the importance of a human-centered system and highlighted the key aspects of designing a successful decision support tool, research is still being conducted on a mathematically rigorous conflict resolution algorithm that can replace air traffic controllers rather than support them. To develop and design a human-centered conflict resolution decision support tool that can generate trustworthy solutions for air traffic controllers to be implemented in real-world operations, acquiring proper knowledge of air traffic controllers is essential. Along these lines, this study demonstrated a method that can reveal characteristics of air traffic controllers control strategies by utilizing historical data. The methodology may be a solution to certain limitations and shortfalls that laboratory experiments have in representing the real environment because the proposed method uses data of air traffic controllers operating during actual work. The method can be used to analyze control strategies for much more complex traffic situations in an objective manner. It enables the analysis of air traffic controller strategies under a variety of different conditions (e.g., weather, sector characteristics, layout of airway, and phase of flight) that exist in real-world operations that may impact controllers behavior. This knowledge concerning how air traffic controllers manage traffic under different situations can be accumulated to be used in designing a reliable decision aid that can ensure safety and support air traffic controllers in a plausible manner.

5.2 Limitations and Future Work

This study presents initial results in the development of an algorithm that can identify air traffic controller (ATC) intervention when expected conflicts are identified. However, it must be noted that the conclusions drawn from the study are limited by the small size of the data set and the immaturity of the heuristic algorithm used to characterize air traffic controller intervention under identified loss of separation. Due to the limited cases of vertical maneuvers and speed changes implemented by the air traffic controllers in resolving expected conflicts, the two parameters could not be considered for the development of the algorithm. Thus, the ATC intervention detection algorithm is limited to identifying cases in which only horizontal maneuvers were implemented for resolving expected conflicts. Also, the data collected for the algorithm was limited to ZOB ARTCC. Future studies need to examine the effect that airspace sectors with different characteristics have on how air traffic controllers implement different control strategies. The results contained in this thesis suggest additional avenues of research that could be used to refine the model.

- Efficient way of collecting and generating trajectory data.

  The process of collecting closed-loop trajectory data from the FlightAware database was a labor-intensive process that required an extensive amount of data compiling, editing, and organizing to be used for analysis. Also, in order to analyze a larger set of data, an easier, less time-consuming method for generating an open-loop trajectory is required.

- Improved method of categorizing operationally significant deviation.

  The trajectories of all the aircraft pairs that were identified to have an encounter in a loss of separation incident were plotted, and the resolution strategies were thoroughly examined to determine whether the deviations detected were operationally significant in resolving the conflict. This also required extensive manual labor that limited the analysis of larger data sets. A rule-based algorithm needs

to be developed that considers multiple parameters (horizontal maneuver, vertical maneuver, and change of speed) to automatically identify and categorize operationally significant deviations so that larger sets of data can be analyzed.

- Analysis of air traffic controller control actions with additional factors that potentially lead to a deviation from the filed path.

  Although this study focused on identifying ATC intervention under identified loss of separations, other factors that could lead to a trajectory deviation need to be analyzed. Added information in the analysis, such as embedding a radar image of the weather, can possibly provide insight into deviations that have not been characterized with consideration of a single factor in the analysis. Also, analysis of different strategies that air traffic controllers implement to control workload, rather than just prevent conflicts, needs to be modeled separately.

- Classification of air traffic controller resolutions strategies to generate rule-based resolutions for decision support tools.

  The final stage of this study should be to classify air traffic controller resolution strategies under different intensity events that led to path deviation and present rule-based resolution strategies that can be implemented in the decision support tools.

LIST OF REFERENCES

LIST OF REFERENCES

Banks, S. B. (2002). Perspectives on the state of modeling and simulating human intent inferencing. In *In: Intent inference for users, teams, and adversaries, AAAI fall symposium, AAAI technical report FS-02-05* (p. 210). The AAAI Press.

Bilimoria, K., Sridhar, B., & Chatterji, G. (1996). Effects of conflict resolution maneuvers and traffic density of free flight. In *Proceedings of AIAA Guidance, Navigation, and Control Conference* (pp. 29–31).

Blom, H. A. P., Bakker, G. J., Everdij, M. H. C., & Van der Park, M. N. J. (2003). Collision risk modeling of air traffic. In *Proceedings of European Control Conference.* IEEE.

Campos, L., & Marques, J. M. G. (2010). On the three-dimensional collision probabilities relevant to ATM. In *Proceedings of 27th International Congress of the Aeronautical Sciences, (ICAS) 2010.*

Dale, W. (2013). *Ask air traffic control: USA's most challenging airports.* Retrieved from `http://www.usatoday.com/story/travel/flights/2013/05/17/famous-airport-approach-landing/2165793/`

D'Arcy, J.-F., & Rocco, P. S. (2001). *Air traffic control specialist decision making and strategic planning-a field survey* (Tech. Rep. No. DOT/FAA/CT-TN01/05). DTIC Document.

Dwyer, J. P., & Landry, S. (2009). Separation assurance and collision avoidance concepts for the next generation air transportation system. *Human Interface and the Management of Information. Information and Interaction*, 748–757.

Endsley, M. R., & Rodgers, M. D. (1994). Situation awareness information requirements analysis for en route air traffic control. In *Proceedings of the human factors and ergonomics society annual meeting* (Vol. 38, p. 7175). SAGE Publications.

Erzberger, H. (2004). Transforming the NAS: the next generation air traffic control system. In *In proceedings of 24th international congress of the aeronautical sciences.*

Erzberger, H., & Paielli, R. A. (2002). Concept for next generation air traffic control system. *Air Traffic Control Quarterly*, *10*(4), 355–378.

FAA, M. (2011). NextGen Implementation Plan. *Washington, DC, Available: http://www. faa. gov/nextgen/media/ng2011_impleme ntation_plan. pdf (accessed March 18, 2011).*

Fothergill, S., & Neal, A. (2008). The effect of workload on conflict decision making strategies in air traffic control. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (Vol. 52, pp. 39–43). Sage Publications.

Gordon, R., Shorrock, S. T., & Pozzi, S. (2005). Predicting and simulating human errors in using the airborne separation assurance. *Human Factors and Aerospace Safety*, *5*(1), 43–60.

Inoue, K., Ando, H., Aoyama, H., & Yamato, H. (2006). A research on task analysis system for enroute air traffic control (PSAM-0254). In *Proceedings of the Eighth International Conference on Probabilistic Safety Assessment & Management (PSAM).* ASME Press.

Inoue, S., Furuta, K., Nakata, K., Kanno, T., Aoyama, H., & Brown, M. (2012). Cognitive process modeling of controllers in en route air traffic control. *Ergonomics*, *55*, 450–464.

Jha, P. D., Bisantz, A. M., & Parasuraman, R. (2003). Through the lens: A new approach to decision modeling under free flight. In *Proceedings of the human factors and ergonomics society annual meeting* (Vol. 47, pp. 349–353). SAGE Publications.

Julier, S. J., & Uhlmann, J. K. (1997). Consistent debiased method for converting between polar and cartesian coordinate systems. In *AeroSense'97* (pp. 110–121). International Society for Optics and Photonics.

Junkins, J. L., & Shuster, M. D. (1993). The geometry of the euler angles. *Journal of the Astronautical Sciences*, *41*, 531–543.

Kallus, K. W., Van Damme, D., & Dittmann, A. (1999). *Integrated task and job analysis of air traffic controllersphase 2: task analysis of en-route controllers* (Tech. Rep. No. HUM.ET1.ST01.1000-REP-04).

Karikawa, D., Takahashi, M., & Aoyama, H. (2010). Performance visualization in air traffic control using cognitive systems simulation. In *Proceedings of 27th Congress of International Council of the Aeronautical Science. (ICAS)* (Vol. 11).

Kim, Y., Lee, S., Lee, K., & Kang, J.-Y. (2013). A development of 3-d resolution algorithm for aircraft collision avoidance. *International Journal of Aeronautical and Space Sciences*, *14*(3), 272–281.

Kirwan, B., & Flynn, M. (2002). *Towards a controller-based conflict resolution toola literature review* (Tech. Rep. No. ASA.01.CORA.2.DEL04-A.LIT).

Kirwan, B., Flynn, M., & Bretigny, F. (2001). Identification of air traffic controller conflict resolution strategies for the CORA (conflict resolution assistant) project. In *Proceeding of 4th USA/Europe Air Traffic Management Research and Development Seminar* (pp. 3–7).

Kosecka, J., Tomlin, C., Pappas, G., & Sastry, S. (1997). Generation of conflict resolution manoeuvres for air traffic management. In *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on* (Vol. 3, pp. 1598–1603). IEEE.

Krozel, J. (2002). Intelligent tracking of aircraft in the national airspace system. In *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit.*

Kuchar, J. K., & Yang, L. C. (2000). A review of conflict detection and resolution modeling methods. *Intelligent Transportation Systems, IEEE Transactions on*, *1*, 179–189.

Landry, S., & Kim, T. (2010). Separation standards under automated separation assurance. In *Proceedings of the 10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference.*

Landry, S. J. (2011). Human centered design in the air traffic control system. *Journal of Intelligent Manufacturing*, *22*(1), 65–72.

Lanier, R. C., & Coppenbarger, R. (2004). Human factors in the design and development of an advanced decision support tool. In *In proceedings of Digital Avionics Systems Conference, 2004. DASC 04. the 23rd* (Vol. 1, p. 5B). IEEE.

Loft, S., Bolland, S., Humphreys, M. S., & Neal, A. (2009). A theory and model of conflict detection in air traffic control: incorporating environmental constraints. *Journal of Experimental Psychology: Applied*, *15*, 106.

McGraw-Hill. (2009). *McGraw-Hill Concise Encyclopedia of Science and Technology* (6th edition ed.). New York: The McGraw-Hill Companies, Inc.

Mohleji, S. C., & Ostwald, P. A. (2003). Future vision of globally harmonized national airspace system with concepts of operations beyond year 2020. In *Digital Avionics Systems Conference, 2003. DASC'03. the 22nd* (Vol. 1, p. 4A). IEEE.

Netjasov, F. (2012). Framework for airspace planning and design based on conflict risk assessment: Part 2: Conflict risk assessment model for airspace tactical planning. *Transportation research part C: emerging technologies*, *24*, 213–226.

Niessen, C., Eyferth, K., & Bierwagen, T. (1999). Modelling cognitive processes of experienced air traffic controllers. *Ergonomics*, *42*, 1507–1520.

Nijhuis, H. (2000). Role of the human in the evolution of ATM (RHEA). *Final Report. NLR: Netherlands*.

Özgür, M., & Cavcar, A. (2008). A knowledge-based conflict resolution tool for en-route air traffic controllers. *Aircraft Engineering and Aerospace Technology*, *80*, 649–656.

Pineda-Krch, M. (2010). *Great-circle distance calculations in r.* Retrieved from `http://www.r-bloggers.com/great-circle-distance-calculations-in-r/`

Prandini, M., Hu, J., Lygeros, J., & Sastry, S. (2000). A probabilistic approach to aircraft conflict detection. *Intelligent Transportation Systems, IEEE Transactions on*, *1*, 199–220.

Prandini, M., & Watkins, O. J. (2005). Probabilistic aircraft conflict detection. *HYBRIDGE, IST-2001*, *32460*.

Rantanen, E. M., & Nunes, A. (2005). Hierarchical conflict detection in air traffic control. *The International Journal of Aviation Psychology*, *15*(4), 339–362.

Rantanen, E. M., & Wickens, C. D. (2012). Conflict resolution maneuvers in air traffic control: Investigation of operational data. *The International Journal of Aviation Psychology*, *22*(3), 266–281.

Rantanen, E. M., Yang, J., & Yin, S. (2006). Comparison of pilots' and controllers' conflict resolution maneuver preferences. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (Vol. 50, pp. 16–19). Sage Publications.

Redding, R. E., Cannon, J. R., & Seamster, T. L. (1992). Expertise in air traffic control (ATC): what is it, and how can we train for it? In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (Vol. 36, pp. 1326–1330). SAGE Publications.

Rong, J., Geng, S., Valasek, J., & Ioerger, T. R. (2002). Air traffic conflict negotiation and resolution using an onboard multi-agent system. In *Digital Avionics Systems Conference, 2002. proceedings. the 21st* (Vol. 2, pp. 7B2–1). IEEE.

Sheridan, T. B., & Parasuraman, R. (2005). Human-automation interaction. *Reviews of Human Factors and Ergonomics*, *1*(1), 89–129.

Stankovic, S., Raufaste, r., & Averty, P. (2008). Determinants of conflict detection: A model of risk judgments in air traffic control. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *50*(1), 121–134.

Vaddi, S., Kwan, J., Fong, A., & Cheng, V. (2012). deterministic and probabilistic conflict detection algorithms for nextgen airport surface operations,. In *Proceedings of the Guidance, Navigation, and Control Conference.*

Vela, A., Clarke, J.-P., Feron, E., Durand, N., & Singhose, W. (2011). Determining the value of information for minimizing controller taskload: a graph-based approach'. In *Ninth USA/Europe Air Traffic Management Research and Development Seminar (ATM 2011).*

Williams, E. (1997). *Aviation formulary.* Retrieved from `http://williams.best.vwh.net/avform.htm`

Zhao, Y., & Schultz, R. (1997). Deterministic resolution of two aircraft conflict in free flight. In *Proceedings of AIAA Guidance, Navigation and Control Conference, AIAA-97* (Vol. 3547).

APPENDICES

APPENDIX A
CONFLICT DETECTION ALGORITHM
(SOURCE CODE)

```
function [cw] = maincomp(m,tr)

% MAINCOMP performs computation for the data and allocate
% information to the base space

DataList = evalin('base','DataList');
DataFolder = evalin('base','DataFolder');
size_data = evalin('base','size_data');
if(tr)% when trajectory
    [m_t m_la m_lo m_al] = maincomp_aircraft(m);
    cw=[];
    for c = 1:size_data
        if(m≠c)
            [c_t c_la c_lo c_al] = maincomp_aircraft(c);
            [mi mf ci cf] = maincomp_range(m_t, c_t);
            if mi ≠ 0
                thereisconflict=maincomp_Tcalc(m,mi,mf,c,ci,cf,m_t,
                                m_la,m_lo,m_al,c_t,c_la,c_lo,c_al);
                if(thereisconflict)
                    cw=[cw,c];
                end
            end
        end
    end
    return;
```

```
elseif(m)
    assignin('base','CONFLICT',zeros(1,9));
    [m_t m_la m_lo m_al] = maincomp_aircraft(m);
    assignin('base','m_t',m_t);
    for c = 1:size_data
        if m ≠ c
            [c_t c_la c_lo c_al] = maincomp_aircraft(c);
            [mi mf ci cf] = maincomp_range(m_t, c_t);
            if mi ≠ 0
                maincomp_calc(m,mi,mf,c,ci,cf,m_t,m_la,m_lo,m_al
                                ,c_t,c_la,c_lo,c_al);
            end
        end
    end
else
    assignin('base','CONFLICT',zeros(1,9));
    [m_t m_la m_lo m_al]=maincomp_aircraft(1);
    ti=m_t(1);
    tf=ti;
    for(m=1:size_data)
        [m_t m_la m_lo m_al]=maincomp_aircraft(m);
        [ti tf]=maincomp_range_all(ti,tf,m_t);
        for(c=m+1:size_data)
            if(m<c)
                [c_t c_la c_lo c_al]=maincomp_aircraft(c);
                [mi mf ci cf]=maincomp_range(m_t, c_t);
                if(mi≠0)
                    maincomp_calc(m,mi,mf,c,ci,cf,m_t,m_la,m_lo,m_al
                                    ,c_t,c_la,c_lo,c_al);
                end
            end
        end
    end
    assignin('base','ti',ti);
    assignin('base','tf',tf);
```

```
61 end
62 end
63
64 function [mi mf ci cf] = maincomp_range(m_t, c_t)
65 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
66 % input: m_t  c_t
67 % output: mi mf ci cf
68 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
69 % this function yields the four indices: main, compared_aircraft,
70 % initial, and final indices
71 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
72
73 mi = 1;
74 ci = 1;
75
76 [mf,g1] = size(m_t);
77 [cf,g2] = size(c_t);
78
79 if m_t(mi) < c_t(ci)
80     while m_t(mi) < c_t(ci)
81         if m_t(mf) < c_t(ci)
82             mi = 0;
83             break
84         end
85         mi = mi + 1;
86     end
87 elseif m_t(mi) > c_t(ci)
88     while m_t(mi) > c_t(ci)
89         if m_t(mi) > c_t(cf)
90             mi = 0;
91             break
92         end
93         ci = ci + 1;
94     end
95 end
```

```
96
97  if m_t(mf) < c_t(cf)
98      while m_t(mf) < c_t(cf)
99          if mi == 0
100             break
101         end
102         cf = cf - 1;
103     end
104 elseif m_t(mf) > c_t(cf)
105     while m_t(mf) > c_t(cf)
106         if mi == 0
107             break
108         end
109         mf = mf - 1;
110     end
111 end
112
113 if mi == 0
114     ci = 0;
115     mf = 0;
116     cf = 0;
117 end
118 end
119
120 function [] = ...
        maincomp_calc(m,mi,mf,c,ci,cf,m_t,m_la,m_lo,m_al,c_t,c_la,
121 c_lo,c_al)
122 R = 6371; % assuming radius of the earth is 6371 km
123 Dlimit = 9260;
124 Alimit = 1000;
125
126 length = mf - mi; % or it can be pf - pi
127
128 tempD = 0; % Distance (difference in latitude and longitude)
129 tempA = 0; % Altitude (difference in altitude)
```

```
130 sw=0;
131
132 for i = 0:1:length
133     lat1 = m_la(mi+i)*pi/180;
134     lat2 = c_la(ci+i)*pi/180;
135     lon1 = m_lo(mi+i)*pi/180;
136     lon2 = c_lo(ci+i)*pi/180;
137     dlat = abs(lat2 - lat1);
138     dlon = abs(lon2 - lon1);
139     a = (sin(dlat/2))^2 + cos(lat1)*cos(lat2)*(sin(dlon/2))^2;
140     cl = 2 * atan2(sqrt(a), sqrt(1 - a));
141     tempD = R*cl*1000; % d in km, mulptiplied by 1000 to get m
142     tempA = abs(m_al(mi+i) - c_al(ci+i));
143     if (tempD < Dlimit) && (tempA < Alimit)
144         % condition to detect conflicts; to be used in numConflicts
145         t=m_t(mi+i);
146         maincomp_conflictinfo(m,c,m_la(mi+i),c_la(ci+i),m_lo(mi+i),
147                 c_lo(ci+i),m_al(mi+i),c_al(ci+i),t);
148     end
149 end
150 end
151
152 function [] = maincomp_conflictinfo(m,c, m_la, c_la, m_lo, c_lo,
153 m_al, c_al, t)
154 % Recall two matrix (one for each main and one for total)
155 MATRIX = evalin('base','CONFLICT');
156 if MATRIX(1,1) == 0
157     MATRIX = [m c m_la c_la m_lo c_lo m_al c_al t];
158 else
159     MATRIX = [MATRIX;[m c m_la c_la m_lo c_lo m_al c_al t]];
160 end
161 assignin('base','CONFLICT',MATRIX);
162 end
163
164 function [ti tf]=maincomp_range_all(ti,tf,m_t)
```

```
165 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
166 % input: ti tf m_t
167 % output: ti tf
168 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
169 % this function is used only when user wants to have "all" cases
170 % for conflict plot and trajectory
171 % this function updates ti and tf
172 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
173
174 [length garb]=size(m_t); % measure the size to get the length //
175 % length will be used to find the greatest timestamp in each main
176 % time array
177 % check two cases: ti being greater than the first timestamp in
178 % each main time array
179 % tf being less than the last timestamp in each main time array
180 if(ti>m_t(1))      % ti check
181     ti=m_t(1);
182 end
183 if(tf<m_t(length)) % tf check
184     tf=m_t(length);
185 end
186 end
187
188 function [thereisconflict] = ...
       maincomp_Tcalc(m,mi,mf,c,ci,cf,m_t,m_la,
189 m_lo,m_al,c_t,c_la,c_lo,c_al)
190 thereisconflict=0;
191 R = 6371; % assuming radius of the earth is 6371 km
192 Dlimit = 9260;
193 Alimit = 1000;
194
195 length = mf - mi; % or it can be pf - pi
196 tempD=0; % Distance (difference in latitude and longitude)
197 tempA=0; % Altitude (difference in altitude)
198
```

```
for i=0:1:length
    lat1 = m_la(mi+i)*pi/180;
    lat2 = c_la(ci+i)*pi/180;
    lon1 = m_lo(mi+i)*pi/180;
    lon2 = c_lo(ci+i)*pi/180;
    dlat = abs(lat2 - lat1);
    dlon = abs(lon2 - lon1);
    a = (sin(dlat/2))^2 + cos(lat1)*cos(lat2)*(sin(dlon/2))^2;
    cl = 2 * atan2(sqrt(a), sqrt(1 - a));
    tempD = R*cl*1000; % d in km, mulptiplied by 1000 to get m
    tempA = abs(m_al(mi+i) - c_al(ci+i));
    if((tempD<Dlimit)&&(tempA<Alimit))
        thereisconflict=1;
        return;
    end
end
end

-------------------------------------------------------------------------------
function [m_t m_la m_lo m_al] = maincomp_aircraft(m)

% MAINCOMP_AIRCRAFT performs assignment of the four arrays
%
%   [m_t m_la m_lo m_al] = maincomp_aircraft(m)
%   Input :  m  - Input aircraft indices from the user from
%                 the gui files. 0 means all aircraft.
%
%   Output: m_t  - time array of the aircraft index directs.
%
%           m_la - latitude array of the aircraft index directs.
%
%           m_lo - longitude array of the aircraft index directs.
%
%           m_al - altitude array of the aircraft index directs.

```

```
234 DataList = evalin('base','DataList');
235 DataFolder = evalin('base','DataFolder');
236 WantedData = strcat(DataFolder,DataList(m));
237 fileID = fopen(WantedData{1});
238 C = textscan(fileID, '%f %f %f %f %*[^\n]' ,'Headerlines',6);
239
240 m_t = C{1};
241 m_la = C{2};
242 m_lo = C{3};
243 m_al = C{4};
244
245 if (m_t(1) ≠ round(m_t(1))) && (m_t(2) ≠ round(m_t(2)))
246     m_la_ori = m_la;
247     m_lo_ori = m_lo;
248     m_al_ori = m_al;
249     [x y] = size(m_t);
250     m_t2 = linspace(round(m_t(1)),round(m_t(x)),round(m_t(x)
251         -round(m_t(1))+1)).';
252     m_la = interp1(m_t,m_la_ori,m_t2,'pchip');
253     m_lo = interp1(m_t,m_lo_ori,m_t2,'pchip');
254     m_al = interp1(m_t,m_al_ori,m_t2,'pchip');
255     m_t = m_t2;
256 end
257 fclose(fileID);
258 end
259
260 ------------------------------------------------------------------------
261 function [] = marker(m,keyhistory)
262
263 % MARKER performs printing information about the conflict with
264 % each time range and a list of corresponding aircraft names.
265 %
266 %   [] = marker(m,keyhistory)
267 %
268 %   Input :      m     - Input aircraft indices from the user
```

```
269 %                           from the gui files. 0 means all aircraft.
270 %
271 %            keyhistory - Conflict matrix that contains the indices
272 %                           of the reference and conflict aircraft and
273 %                           time and either having conflict or leaving
274 %                           conflict. The sorted order is as followed:
275 %                           conflict, main,time, add/subtract.
276
277 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
278 %                                KEY matrix maker                       %
279 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
280 DataList=evalin('base','DataList');
281 [keylength garb2]=size(keyhistory);
282 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
283 %                              marker matrix build                      %
284 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
285 j=1;                                          % pointer for key history
286 cnt=0;                                        % mrkstr size
287 k=1;                                          % pointer for mrkstr
288 mrkstr=[];
289 while(j<keylength)
290     air1=keyhistory(j,1);
291     air2=keyhistory(j,3);
292
293 %%%%%%%%%%%%%%%%%%%%%%%%%% NO REPETITION  %%%%%%%%%%%%%%%%%%%%%%%%%%
294
295     if(keyhistory(j,2)≠keyhistory(j+1,2))
296         if(j==1) % first time?
297             mrkstr=[keyhistory(1,1) keyhistory(1,3)];
298                         % first row is always adding
299             cnt=1;
300             j=2;
301         else % not the first time?
302             if(keyhistory(j,4))         % if add?
303                 mrkstr=[mrkstr;[air1,air2]];
```

```
304                     % use strcat to add the issued string
305                 cnt=cnt+1;
306             else  % if subtract?
307                 for k=1:cnt
308                     postit=ismember(mrkstr(k,:),[air1 air2]);
309                                 % search "mrkstr"
310                     if((postit(1)==1)&&(postit(2)==1))
311                                 % find the row
312                         mrkstr(k,1)=0;
313                         mrkstr(k,2)=0;
314                     end
315                 end
316             end
317             j=j+1;                  % update the current pointer
318         end
319
320 %%%%%%%%%%%%%%%%%%%%%%%%%% REPETITION %%%%%%%%%%%%%%%%%%%%%%%%%%%%
321
322     else
323         tempj=j;
324 % temporary pointer to read array till the end of repetition
325         tempval=keyhistory(j,2);
326 % create temporary value to check till when it repeats
327         while((tempj≤keylength)&&(tempval==keyhistory(tempj,2)))
328             air1=keyhistory(tempj,1);
329 % aircraft differs in each loop//
330 % it is the aircraft name that we have to add
331             air2=keyhistory(tempj,3);
332             if(keyhistory(tempj,4)) % if add?
333                 mrkstr=[mrkstr;[air1,air2]];
334                 cnt=cnt+1;
335             else                    % if subtract?
336                 for k=1:cnt  % max pointer
337                     postit=ismember(mrkstr(k,:),[air1 air2]);
338                                     % search "mrkstr"
```

```
339                     if((postit(1)==1)&&(postit(2)==1))
340                                             % find the row
341                         mrkstr(k,1)=0;
342                         mrkstr(k,2)=0;
343                     end
344                 end
345             end
346             tempj=tempj+1;
347         end
348         if(tempj>keylength)
349             j=keylength;
350         else
351             j=tempj;
352 % update current pointer to end of repetition
353         end
354
355 %%%%%%%%%%%%%%%%%%%%%%%%%%%% IF ENDS %%%%%%%%%%%%%%%%%%%%%%%%%%%%
356
357     end
358     marker_print(m,keyhistory(j-1,2),keyhistory(j,2),mrkstr,cnt);
359 end
360 end
361
362 function []=marker_print(m,ti,tf,mrkstr,length)
363 fprintf('Time[%d %d] => ',ti,tf);
364 % print the issued aircraft in each region
365 DataList=evalin('base','DataList');
366 i=1;
367 DataList=evalin('base','DataList');
368 if(m)
369 for i=1:length
370     if(mrkstr(i,1))
371         fprintf('%s  ',strrep(DataList{mrkstr(i,1)},'.txt',''));
372     end
373 end
```

```
374 else
375     for i=1:length
376         if(mrkstr(i,1))
377             fprintf('(%s with %s) ',
378                         strrep(DataList{mrkstr(i,1)},'.txt',''),
379                         strrep(DataList{mrkstr(i,2)},'.txt',''));
380         end
381     end
382 end
383 fprintf('\n');
384 end
385
386 ---------------------------------------------------------------------------
387
388 function[]=plotter(m)
389
390 % PLOTTER performs plot of the number of conflicts at each time
391 %
392 %   []=plotter(m)
393
394 keyhistory=plotter_matrix(m);
395 plotter_plot(m,keyhistory);
396 end
397
398 function [keyhistory] = plotter_matrix(m)
399 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
400 % input: none                                                            %
401 % output: soakedmatrix                                                   %
402 %         later in the future it will have a name "key"                  %
403 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
404 CONFLICT = evalin('base','CONFLICT');
405 key = [CONFLICT(:,1) CONFLICT(:,2) CONFLICT(:,9)];
406 conflicttime=key(:,3);
407 conflictwith1=key(:,1);
408 conflictwith2=key(:,2);
```

```
409 [endpt garb1] = size(key);
410 previous = 1;
411 current = 2;
412 sw = 0;
413 % boolean logic (TRUE or FALSE)
414 soakedmatrix=zeros(1,4);
415 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
416 %                    comp ti tf main matrix begins                    %
417 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
418 while(current ≤ endpt)
419     if(conflicttime(previous)+1 ≠ conflicttime(current))
420         if(sw) %there is a value
421             soakedmatrix=[soakedmatrix ; ...
                    [conflictwith2(previous) sw
422                             conflicttime(previous) ...
                                conflictwith1(previous)]];
423             sw=conflicttime(current);
424         else
425             soakedmatrix = [conflictwith2(previous) conflicttime(1)
426                             conflicttime(previous) ...
                                conflictwith1(previous)];
427             sw = conflicttime(current);
428         end
429     end
430     previous=previous+1;
431     current=current+1;
432 end
433 soakedmatrix=vertcat(soakedmatrix,[conflictwith2(previous)
434 sw conflicttime(previous) conflictwith1(previous)]);
435 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
436 %                      comp ti tf main matrix ends                    %
437 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
438 [keylength garb2]=size(soakedmatrix);
439 clear garb1 garb2 sw previous current endpt; % memory clear!
440 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
441 %                          comp t main +/- begins                    %
442 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
443 ascend=[soakedmatrix(:,1) soakedmatrix(:,2) soakedmatrix(:,4)];
444 descend=[soakedmatrix(:,1) soakedmatrix(:,3) soakedmatrix(:,4)];
445 % left is conflict name & right is time
446 add=[ascend;descend];
447 keyhistory=zeros(2*keylength,1);   % half of the array is 1
448 for i=1:1:keylength
449     keyhistory(i)=1;
450 end                                % rest half is 0
451 keyhistory=[add keyhistory];
452 % ascend has 1, descend has 0, the matrix is sorted by time
453 keyhistory=sortrows(keyhistory,2); % sort by time
454 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
455 %                           comp t main +/- ends                     %
456 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
457 assignin('base','keyhistory',keyhistory);
458 end
459
460 function []=plotter_plot(m,keyhistory)
461 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
462 % input
463 % loop
464 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
465 [keylength garb1]=size(keyhistory);
466 if(m)
467     m_t=evalin('base','m_t');
468 else
469     ti=evalin('base','ti');
470     tf=evalin('base','tf');
471     m_t = ti:tf;                % call the array m_t
472 end
473 if(m)
474     [mlength garb1]=size(m_t);
475         % have to know length of m_t to make Y-axis of the plot
```

```
476 else
477     [garb1 mlength]=size(m_t);
478 end
479 clear garb1;
480 numconflictarray = zeros(mlength,1); % matrix for Y-axis of the plot
481 i=1;  % pointer for m_t array
482 j=1;  % pointer for keyhistory matrix
483 sw=0; % boolean logic true or false
484 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
485 % FILL UP
486 % loop
487 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
488 while(j<keylength) % first loop
489         if(m_t(i)==keyhistory(j,2))
490                 % if "m_t" meets "keyhistory" time elements
491             if(keyhistory(j,2)≠keyhistory(j+1,2))
492                         % no repetition
493                 if(keyhistory(j,4)) % if switch is on
494                     sw=sw+1; % add
495                 else                % otherwise
496                     sw=sw-1; % 0
497                 end
498                 j=j+1;
499             else                    % repetition
500                 tempj=j;
501                 tempval=keyhistory(tempj,2);
502             while((tempj≤keylength)&&(keyhistory(tempj,2)==tempval))
503                     if(keyhistory(tempj,4))     % if switch is on
504                         sw=sw+1;           % add 1
505                     else                % otherwise
506                         sw=sw-1;           % 0
507                     end
508                     tempj=tempj+1;
509                 end
510                 j=tempj;
```

```
511             end
512         end
513         numconflictarray(i)=sw;
514                 % index for m_t and numconflictarray are the same
515         i=i+1;
516 end
517 while(i≤mlength) % second loop
518     if(m_t(i)==keyhistory(j,2))
519         % if "m_t" meets "keyhistory" time elements
520         if(keyhistory(j,4))    % if switch is on
521             sw=sw+1;            % add 1
522         else                % otherwise
523             sw=sw-1;            % 0
524         end
525     end
526     numconflictarray(i)=sw;
527     i=i+1;
528 end
529 if(m)
530     m_t=[m_t(1)-1;m_t];     % before the main aircraft enters,
531 else
532     m_t=[m_t(1)-1 m_t];
533 end
534 numconflictarray=[0;numconflictarray]; % the value is always 0
535 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
536 plot(m_t,numconflictarray);
537 title('The Set of Conflicts that occur for each Timestamp');
538 xlabel('timestamp (second)');
539 ylabel('Number of conflicts that the main aircraft encounter ...
        (number)');
540 end
541
542 ---------------------------------------------------------------------------
543 function[]=showlist(cw)
544
```

```
% SHOWLIST performs collection of the aircraft that has conflict
%          with and ask users to put which aircraft that the users
%          are interested in plotting the trajectory
%
%   []=showlist(cw)
%   Input : cw - The indices of aircraft that has conflict with the
%                reference aircraft.

[garb1 cwlength]=size(cw);
constructlength=cwlength;
% rebuild length --> will be modified as cw is reduced
for i=1:cwlength-1
% read from the first element to the 2nd last element
    if(cw(i))      % if element is not 0
        temp=cw(i);% save whatever value
        for k=i+1:cwlength
% read from the next element to the end
            if(cw(k)==temp)
                cw(k)=0;
                constructlength=constructlength-1;
% reduce the size of the construct length
            end
        end
    end
end
cw_new=zeros(constructlength,1); % build an empty array
p=1;
for j=1:cwlength
    if(cw(j))            % only when cw element is not zero
        cw_new(p)=cw(j); % save it to the new
        p=p+1;
    end
end

DataList=evalin('base','DataList');
```

```
580 [clength garb2]=size(cw_new);
581 clear garb1 garb2;
582 fprintf('Compared aircraft(s) with confliction is(are):\n')
583 for n = 1:clength
584     fprintf('%d. %s\n',n,strrep(DataList{cw_new(n)},'.txt',''))
585 end
586 fprintf('\nTo plot, select numbers from the list.
587 If ALL aircraft plot is desired, enter 0.\n');
588 fprintf('If MULTIPLE aircraft plots are desired,
589 separate aircraft numbers with comma.');
590 fprintf('\n(For example, 1,2,3 for aircraft 1, 2 and 3.)\n');
591 end
592
593 ------------------------------------------------------------------------
594
595 function[]=trajectory(m,comp_input,cw)
596
597 % TRAJECTORY performs plot of the trajectory of the reference
598 % and conflict aircraft that the user is interested in
599 %
600 %   []=trajectory(m,comp_input,cw)
601
602 comp=str2num(comp_input);
603 [garb clength]=size(comp);
604 cw_new=zeros(clength,1);
605 for n = 1:clength
606     cw_new(n)=cw(comp(n));
607 end
608
609 DataList = evalin('base','DataList');
610 [m_t m_la m_lo m_al] = maincomp_aircraft(m);
611 M = [m_lo m_la m_al.*.3048]; %long, lat, al (deg, m)
612 M_cart = ell2cart(M);
613 plot(M_cart(:,1),M_cart(:,2),
614 'DisplayName',strrep(DataList{m},'.txt',''));
```

```
615 legend('-DynamicLegend');
616 hold all;
617 for i=1:clength
618     clear m_t m_la m_lo m_al M M_cart;
619     [m_t m_la m_lo m_al] = maincomp_aircraft(cw_new(i));
620     M = [m_lo m_la m_al.*.3048]; %long, lat, al (deg, m)
621     M_cart = ell2cart(M);
622     plot(M_cart(:,1),M_cart(:,2),'DisplayName',
623         strrep(DataList{cw_new(i)},'.txt',''));
624 end
625 end
626
627 ------------------------------------------------------------------------
628
629 function [m]=getData(m_string)
630
631 % GETDATA performs allocation of datalist as an array and user input
632 %
633 %   [m] = getData(m_string)
634 %
635 %   Input : m_string - Input from the user from the gui files.
636 %
637 %
638 %   Allocation: DataList - It is an array of names of aircraft
639 %                          saved as textfiles in another folder.
640 %
641 %                extension - 'txt.'.
642 %
643 %   Output:   m - The index number of the main aircraft from the
644 %                 user's input. When user asks for total conflict
645 %                 flight, m is saved as 0.
646
647 DataFolder='../Data/';
648 extension='.txt';
649 assignin('base','DataFolder',DataFolder);
```

```
650 assignin('base','extension',extension);
651 m=getData_name_finder(m_string); %getting data list, find main
652 DataList=evalin('base','DataList');
653 size_data_mat=size(DataList); %getting size of data
654 size_data=size_data_mat(2);
655 clear size_data_mat;
656 assignin('base','size_data',size_data);
657 assignin('base','m',m);
658 end
659
660 function [m]=getData_name_finder(m_string)
661 m=-1;
662 sw=1;
663 while(m<0)
664     if(strcmp(m_string,'all'))
665         m=0;
666     end
667     getData_name_extractor;
668     datalist = evalin('base','DataList');
669     extension = evalin('base','extension');
670     datalocation = strcat(m_string, extension);
671     if(find(ismember(datalist,datalocation)))
672         m=find(ismember(datalist,datalocation));
673     end
674     if(m<0)
675         fprintf('\nwe are sorry. the program failed to find
676                 the aircraft from the directory.\nplease check the
677                 directory and type in the main aircraft
678                 viewpoint again.\n');
679         fprintf('\nby default, the program will run the
680                 first aircraft in the list.\n\n');
681         m=1;
682     end
683 end
684 end
```

```
685
686 function[] = getData_name_extractor
687     DataFolder = evalin('base','DataFolder');
688     extension = evalin('base','extension');
689     Datalocation = strcat(DataFolder,'*',extension);
690     a = dir(Datalocation);
691     assignin('base','DataList',{a.name});
692 end
693
694 ---------------------------------------------------------------------
695
696 function CART=ell2cart(ELL,ellips,FileOut)
697
698 % ELL2CART performs transformation from ellipsoidal coordinates
699 % to cartesian coordinates
700 %
701 % CART=ell2cart(ELL,ellips,FileOut)
702 %
703 % Inputs:   ELL  Geographic coordinates as nx3-matrix
704 % (longitude, latitude, height) [degree, m]
705 % 3xn-matrices are allowed. Be careful with 3x3-matrices!
706 % nx2-matrices are allowed, all heights are set to 0 in that case.
707 % Southern hemisphere is signaled by negative latitude.
708 % ELL may also be a file name with ASCII data to be processed.
709 % No point IDs, only coordinates as if it was a matrix.
710 %
711 % The underlying ellipsoid as string in lower case letters,
712 % default if omitted or set to [] is 'besseldhdn'
713 % See Ellipsoids.m for details.
714 %
715 % Outputs: CART  nx3-matrix with right-handed
716 % cartesian coordinates (x y z) in [m]
717
718 % Author:
719 % Peter Wasmeier, Technical University of Munich
```

```
720 % p.wasmeier@bv.tum.de
721 % Jan 18, 2006
722
723 %% Do some input checking
724
725 % Load input file if specified
726 if ischar(ELL)
727     ELL=load(ELL);
728 end
729
730 % Input size checking and defaults
731 if ¬any(ismember(size(ELL),[2 3]))
732     error('Coordinate list ELL must be a nx3- or nx2-matrix!')
733 elseif (ismember(size(ELL,1),[2 3]))&&(¬ismember(size(ELL,2),[2 3]))
734     ELL=ELL';
735 end
736 if size(ELL,2)==2
737     ELL(:,3)=zeros(size(ELL,1),1);
738 end
739 if nargin<3
740     FileOut=[];
741 end
742 if nargin<2 || isempty(ellips)
743     ellips='besseldhdn';
744 end
745
746 %% Load ellipsoids
747 load Ellipsoids;
748 if ¬exist(ellips,'var')
749     error(['Ellipsoid ',ellips,' is not defined in
750         Ellipsoids.mat - check your definitions!.'])
751 end
752 eval(['ell=',ellips,';']);
753
754 %% Do calculations
```

```
755 CART=zeros(size(ELL));
756 rho=180/pi;
757 B=ELL(:,2)/rho;
758 L=ELL(:,1)/rho;
759
760 % 1. numerical eccentricity
761 e2=(ell.a^2-ell.b^2)/ell.a^2;
762
763 % norm radius
764 N=ell.a./sqrt(1-e2*sin(B).^2);
765
766 % cartesian coordinates
767 CART(:,1)=(N+ELL(:,3)).*cos(B).*cos(L);
768 CART(:,2)=(N+ELL(:,3)).*cos(B).*sin(L);
769 CART(:,3)=(N.*(1-e2)+ELL(:,3)).*sin(B);
770
771 %% Write output to file if specified
772
773 if ~isempty(FileOut)
774     fid=fopen(FileOut,'w+');
775     fprintf(fid,'%12.6f  %12.6f  %12.6f\n',CART');
776     fclose(fid);
777 end
778
779 ---------------------------------------------------------------------
780
781 function varargout = gui_Directory(varargin)
782 % GUI_DIRECTORY MATLAB code for gui_Directory.fig
783
784 gui_Singleton = 1;
785 gui_State = struct('gui_Name',       mfilename, ...
786                    'gui_Singleton',  gui_Singleton, ...
787                    'gui_OpeningFcn', @gui_Directory_OpeningFcn, ...
788                    'gui_OutputFcn',  @gui_Directory_OutputFcn, ...
789                    'gui_LayoutFcn',  [] , ...
```

```
790                     'gui_Callback',   []);
791 if nargin && ischar(varargin{1})
792     gui_State.gui_Callback = str2func(varargin{1});
793 end
794
795 if nargout
796     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
797 else
798     gui_mainfcn(gui_State, varargin{:});
799 end
800
801 % --- Executes just before gui_Directory is made visible.
802 function gui_Directory_OpeningFcn(hObject, eventdata,
803 handles, varargin)
804 % This function has no output args, see OutputFcn.
805 % hObject    handle to figure
806 % eventdata  reserved - to be defined in a future version of MATLAB
807 % handles    structure with handles and user data (see GUIDATA)
808 % varargin   command line arguments to gui_Directory (see VARARGIN)
809
810 % Choose default command line output for gui_Directory
811 handles.output = hObject;
812 DataFolder = '../Data/';
813 extension = '.txt';
814 %Pulling list of names
815 Datalocation = strcat(DataFolder,'*',extension);
816 a = dir(Datalocation);
817
818 %To global workspace
819 assignin('base','DataList',{a.name});
820 DataList=evalin('base','DataList');
821 size_data_mat = size(DataList);
822 size_data = size_data_mat(2);
823 clear size_data_mat;
824 list='';
```

```
825 for i=1:size_data
826     aircraft=strrep(DataList(i),'.txt',' ');
827     list=strcat(list,aircraft);
828     set(handles.edit1,'String',list);
829 end
830 % Update handles structure
831 guidata(hObject, handles);
832 
833 % UIWAIT makes gui_Directory wait for user response (see UIRESUME)
834 % uiwait(handles.figure1);
835 
836 
837 % --- Outputs from this function are returned to the command line.
838 function varargout = gui_Directory_OutputFcn(hObject, eventdata,
839 handles)
840 % varargout  cell array for returning output args (see VARARGOUT);
841 % hObject    handle to figure
842 % eventdata  reserved - to be defined in a future version of MATLAB
843 % handles    structure with handles and user data (see GUIDATA)
844 
845 % Get default command line output from handles structure
846 varargout{1} = handles.output;
847 
848 function edit1_Callback(hObject, eventdata, handles)
849 % hObject    handle to edit1 (see GCBO)
850 % eventdata  reserved - to be defined in a future version of MATLAB
851 % handles    structure with handles and user data (see GUIDATA)
852 % --- Executes during object creation, after setting all properties.
853 
854 function edit1_CreateFcn(hObject, eventdata, handles)
855 
856 if ispc && isequal(get(hObject,'BackgroundColor'),
857 get(0,'defaultUicontrolBackgroundColor'))
858     set(hObject,'BackgroundColor','white');
859 end
```

```
860
861 ------------------------------------------------------------------------
862
863 function varargout = gui_Executive(varargin)
864 % GUI_EXECUTIVE MATLAB code for gui_Executive.fig
865
866 gui_Singleton = 1;
867 gui_State = struct('gui_Name',       mfilename, ...
868                    'gui_Singleton',  gui_Singleton, ...
869                    'gui_OpeningFcn', @gui_Executive_OpeningFcn, ...
870                    'gui_OutputFcn',  @gui_Executive_OutputFcn, ...
871                    'gui_LayoutFcn',  [] , ...
872                    'gui_Callback',   []);
873 if nargin && ischar(varargin{1})
874     gui_State.gui_Callback = str2func(varargin{1});
875 end
876
877 if nargout
878     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
879 else
880     gui_mainfcn(gui_State, varargin{:});
881 end
882 % End initialization code - DO NOT EDIT
883
884 % --- Executes just before gui_Executive is made visible.
885 function gui_Executive_OpeningFcn(hObject, eventdata,
886 handles, varargin)
887 % This function has no output args, see OutputFcn.
888 % hObject    handle to figure
889 % eventdata  reserved - to be defined in a future version of MATLAB
890 % handles    structure with handles and user data (see GUIDATA)
891 % varargin   command line arguments to gui_Executive (see VARARGIN)
892
893 % Choose default command line output for gui_Executive
894 handles.output = hObject;
```

```
895 handles.check1=0;
896 handles.check2=0;
897 % Update handles structure
898 guidata(hObject, handles);
899
900 % UIWAIT makes gui_Executive wait for user response (see UIRESUME)
901 % uiwait(handles.figure1);
902
903 % --- Outputs from this function are returned to the command line.
904 function varargout = gui_Executive_OutputFcn(hObject, eventdata,
905 handles)
906 % varargout  cell array for returning output args (see VARARGOUT);
907 % hObject    handle to figure
908 % eventdata  reserved - to be defined in a future version of MATLAB
909 % handles    structure with handles and user data (see GUIDATA)
910
911 % Get default command line output from handles structure
912 varargout{1} = handles.output;
913
914 % --- Executes on button press in checkbox_conflict.
915 function checkbox_conflict_Callback(hObject, eventdata,
916 handles)
917
918 handles.check1=get(hObject,'Value');
919 guidata(hObject, handles);
920
921 % --- Executes on button press in checkbox_trajectory.
922 function checkbox_trajectory_Callback(hObject, eventdata, handles)
923
924 % checkbox_trajectory
925 handles.check2=get(hObject,'Value');
926 guidata(hObject, handles);
927
928 % --- Executes on button press in pushbutton_okay.
929 function pushbutton_okay_Callback(hObject, eventdata, handles)
```

```
930
931 handles.check3 = get(hObject,'Value');
932 if((handles.check3)&&(handles.check1)&&(handles.check2))
933     gui_FlightConflict;
934     gui_Trajectory;
935     close gui_Executive;
936 elseif((handles.check3)&&(handles.check1))
937     gui_FlightConflict;
938     close gui_Executive;
939 elseif((handles.check3)&&(handles.check2))
940     gui_Trajectory;
941     close gui_Executive;
942 end
943
944 ------------------------------------------------------------------------
945
946 function varargout = gui_FlightConflict(varargin)
947 % GUI_FLIGHTCONFLICT MATLAB code for gui_FlightConflict.fig
948
949 gui_Singleton = 1;
950 gui_State = struct('gui_Name',       mfilename, ...
951                    'gui_Singleton',  gui_Singleton, ...
952                    'gui_OpeningFcn', ...
                          @gui_FlightConflict_OpeningFcn, ...
953                    'gui_OutputFcn',  ...
                          @gui_FlightConflict_OutputFcn, ...
954                    'gui_LayoutFcn',  [] , ...
955                    'gui_Callback',   []);
956 if nargin && ischar(varargin{1})
957     gui_State.gui_Callback = str2func(varargin{1});
958 end
959
960 if nargout
961     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
962 else
```

```
963     gui_mainfcn(gui_State, varargin{:});
964 end
965 % End initialization code - DO NOT EDIT
966
967 % --- Executes just before gui_FlightConflict is made visible.
968 function gui_FlightConflict_OpeningFcn(hObject, eventdata,
969 handles, varargin)
970 % This function has no output args, see OutputFcn.
971 % hObject    handle to figure
972 % eventdata  reserved - to be defined in a future version of MATLAB
973 % handles    structure with handles and user data (see GUIDATA)
974 % varargin   command line arguments to gui_FlightConflict (see ...
       VARARGIN)
975
976 % Choose default command line output for gui_FlightConflict
977 handles.output = hObject;
978
979 % Update handles structure
980 guidata(hObject, handles);
981
982 % --- Outputs from this function are returned to the command line.
983 function varargout = gui_FlightConflict_OutputFcn(hObject,
984 eventdata, handles)
985
986 varargout{1} = handles.output;
987
988 function edit_input_Callback(hObject, eventdata, handles)
989
990 handles.str = get(hObject,'String');
991 guidata(hObject, handles)
992
993 % --- Executes during object creation, after setting all properties.
994 function edit_input_CreateFcn(hObject, eventdata, handles)
995
996 if ispc && isequal(get(hObject,'BackgroundColor'),
```

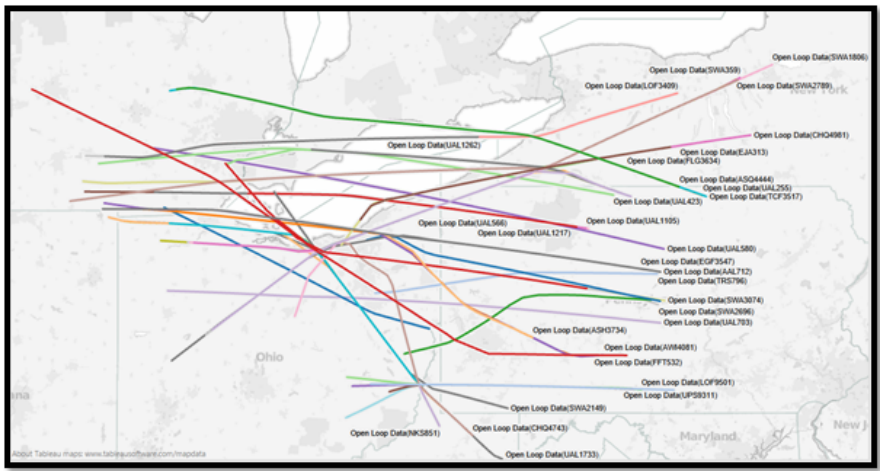
```
997 get(0,'defaultUicontrolBackgroundColor'))
998     set(hObject,'BackgroundColor','white');
999 end
1000
1001 % --- Executes on button press in pushbutton_okay.
1002 function pushbutton_okay_Callback(hObject, eventdata, handles)
1003
1004 handles.check1 = get(hObject,'Value');
1005 if(handles.check1)
1006     m=getData(handles.str);
1007     maincomp(m,0);
1008     plotter(m);
1009 end
1010 guidata(hObject,handles);
1011
1012 function edit_marker_Callback(hObject, eventdata, handles)
1013
1014 % --- Executes during object creation, after setting all properties.
1015 function edit_marker_CreateFcn(hObject, eventdata, handles)
1016
1017 if ispc && isequal(get(hObject,'BackgroundColor'),
1018 get(0,'defaultUicontrolBackgroundColor'))
1019     set(hObject,'BackgroundColor','white');
1020 end
1021
1022 % --- Executes on button press in pushbutton_marker.
1023 function pushbutton_marker_Callback(hObject, eventdata, handles)
1024
1025 check2 = get(hObject,'Value');
1026 keyhistory=evalin('base','keyhistory');
1027 m=evalin('base','m');
1028 if(check2)
1029     marker(m,keyhistory);
1030 end
1031
```

```
1032 % --- Executes on button press in pushbutton_directory.
1033 function pushbutton_directory_Callback(hObject, eventdata, handles)
1034
1035 handles.check2 = get(hObject,'Value');
1036 if(handles.check2)
1037     gui_Directory;
1038 end
1039
1040 ------------------------------------------------------------------------
1041
1042 function varargout = gui_Trajectory(varargin)
1043 % GUI_TRAJECTORY MATLAB code for gui_Trajectory.fig
1044
1045 gui_Singleton = 1;
1046 gui_State = struct('gui_Name',       mfilename, ...
1047                    'gui_Singleton',  gui_Singleton, ...
1048                    'gui_OpeningFcn', @gui_Trajectory_OpeningFcn, ...
1049                    'gui_OutputFcn',  @gui_Trajectory_OutputFcn, ...
1050                    'gui_LayoutFcn',  [] , ...
1051                    'gui_Callback',   []);
1052 if nargin && ischar(varargin{1})
1053     gui_State.gui_Callback = str2func(varargin{1});
1054 end
1055
1056 if nargout
1057     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
1058 else
1059     gui_mainfcn(gui_State, varargin{:});
1060 end
1061
1062 % --- Executes just before gui_Trajectory is made visible.
1063 function gui_Trajectory_OpeningFcn(hObject, eventdata,
1064 handles, varargin)
1065
1066 % Choose default command line output for gui_Trajectory
```

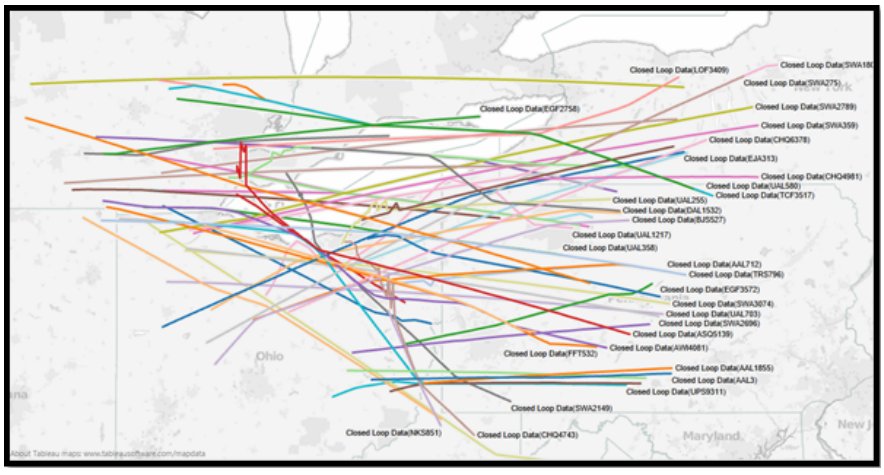
```
1067 handles.output = hObject;
1068
1069 % Update handles structure
1070 guidata(hObject, handles);
1071
1072 % --- Outputs from this function are returned to the command line.
1073 function varargout = gui_Trajectory_OutputFcn(hObject,
1074 eventdata, handles)
1075
1076 varargout{1} = handles.output;
1077
1078 % --- Executes on button press in pushbutton_directory.
1079 function pushbutton_directory_Callback(hObject, eventdata, handles)
1080
1081 handles.check3 = get(hObject,'Value');
1082 if(handles.check3)
1083     gui_Directory;
1084 end
1085
1086 function edit_main_Callback(hObject, eventdata, handles)
1087
1088 handles.str1 = get(hObject,'String');
1089 guidata(hObject, handles);
1090
1091 % --- Executes during object creation, after setting all properties.
1092 function edit_main_CreateFcn(hObject, eventdata, handles)
1093
1094 if ispc && isequal(get(hObject,'BackgroundColor'),
1095  get(0,'defaultUicontrolBackgroundColor'))
1096     set(hObject,'BackgroundColor','white');
1097 end
1098
1099 % --- Executes on button press in pushbutton_showlist.
1100 function pushbutton_showlist_Callback(hObject, eventdata, handles)
1101
```

```
1102 handles.check1 = get(hObject,'Value');
1103 if(handles.check1)
1104     m=getData(handles.str1);
1105     cw=maincomp(m,1);
1106     showlist(cw);
1107     handles.cw=cw;
1108 end
1109 guidata(hObject,handles);
1110
1111
1112 function edit_comp_Callback(hObject, eventdata, handles)
1113
1114 handles.str2 = get(hObject,'String');
1115
1116 guidata(hObject, handles);
1117
1118 % --- Executes during object creation, after setting all properties.
1119 function edit_comp_CreateFcn(hObject, eventdata, handles)
1120
1121 if ispc && isequal(get(hObject,'BackgroundColor'),
1122  get(0,'defaultUicontrolBackgroundColor'))
1123     set(hObject,'BackgroundColor','white');
1124 end
1125
1126 % --- Executes on button press in pushbutton_plot.
1127 function pushbutton_plot_Callback(hObject, eventdata, handles)
1128
1129 handles.check2 = get(hObject,'Value');
1130 m=evalin('base','m');
1131 if(handles.check2)
1132     trajectory(m,handles.str2,handles.cw);
1133 end
```

APPENDIX B
FLIGHT TRAJECTORIES ENCOUNTERING IN LOSS-OF-SEPARATION

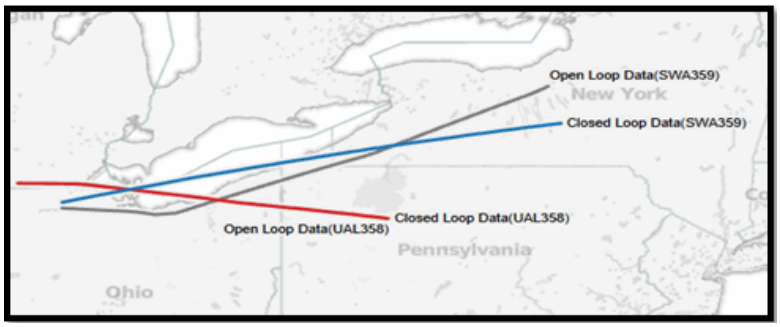


(a) Open-loop trajectories with loss-of-separation encounters.

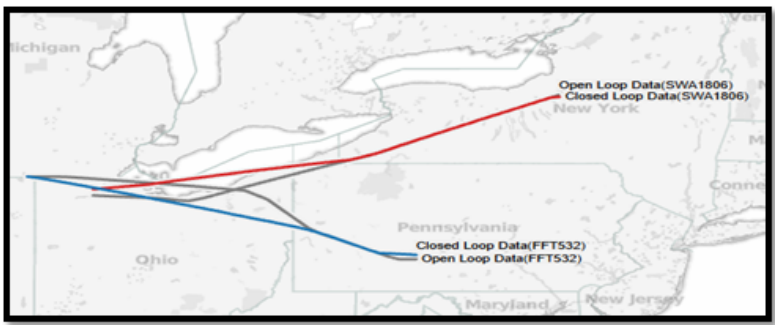


(b) Closed-loop trajectories in correspondence with Figure (a).

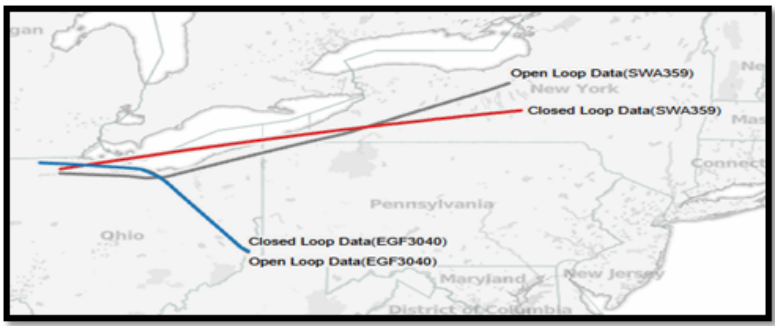
Figure B.1. Comparison of open- and closed-loop trajectories.



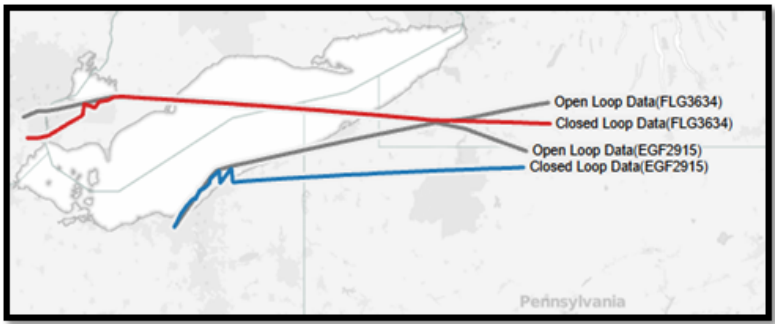
(a) SWA359-UAL358



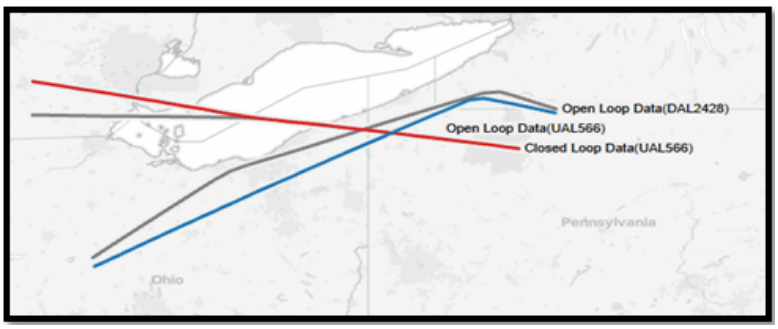
(b) SWA1806-FFT532



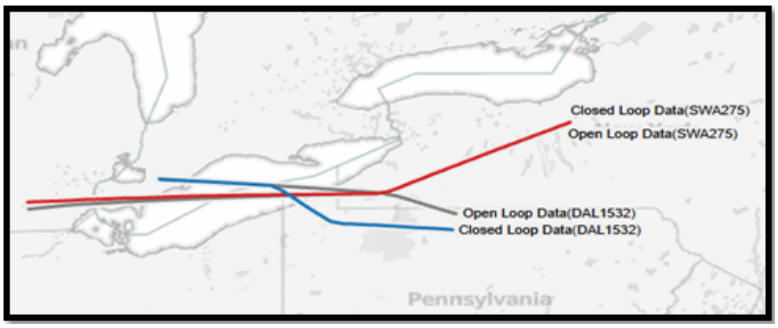
(c) EGF3040-SWA359



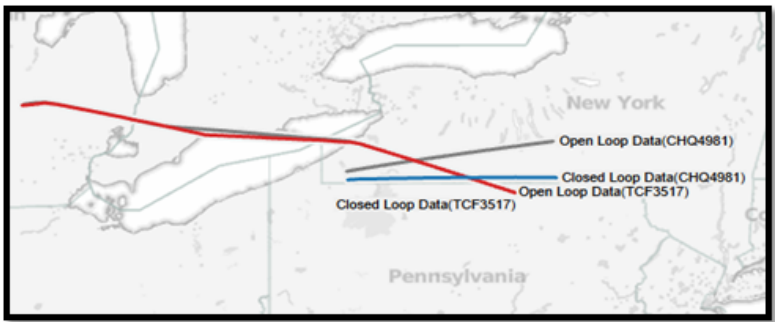
(d) FLG3634-EGF2915



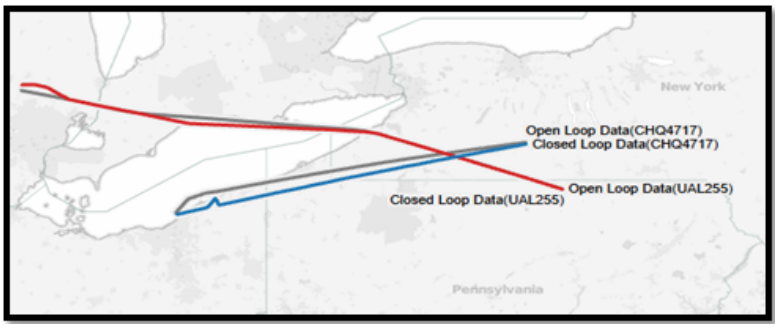
(e) UAL566-DAL2428



(f) DAL1532-SWA275

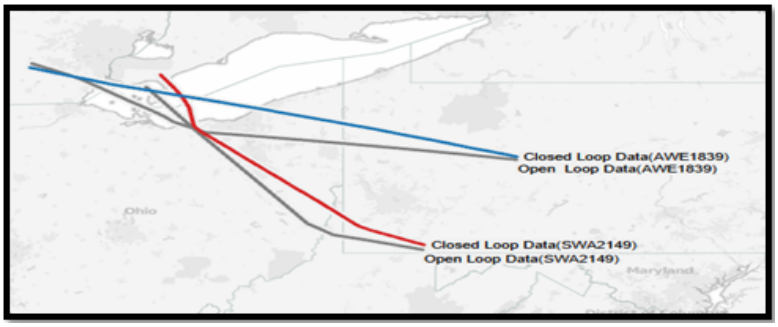


(g) TCF3517-CHQ4981

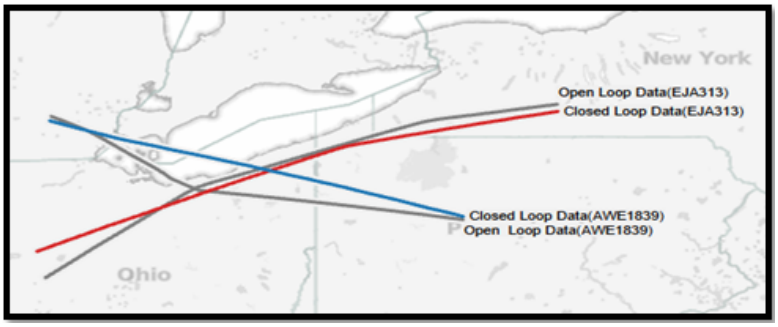


(h) CHQ4717-UAL255

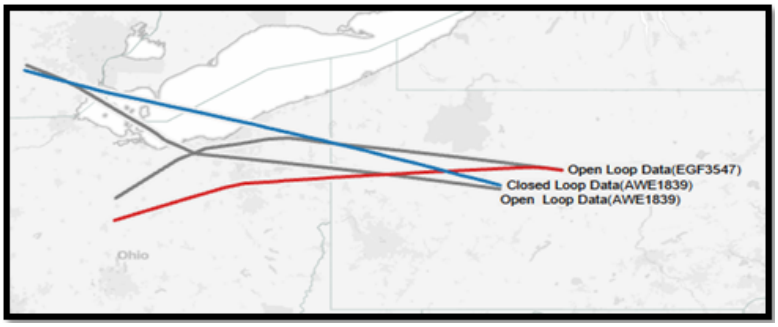
(i) AWE1839-UAL566



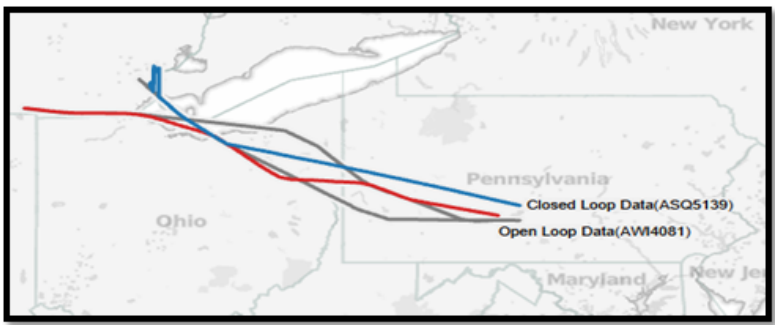
(j) SWA2149-AWE1839



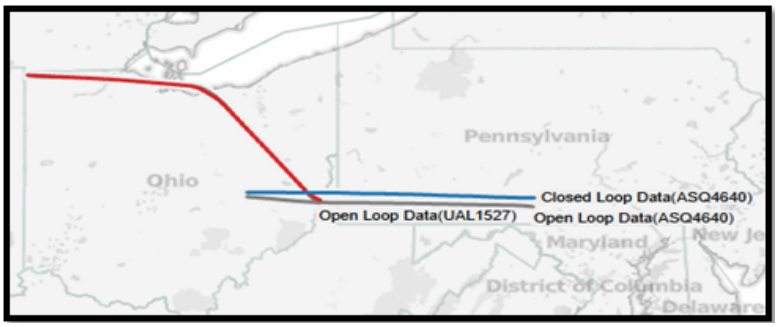
(k) EJA313-AWE1839



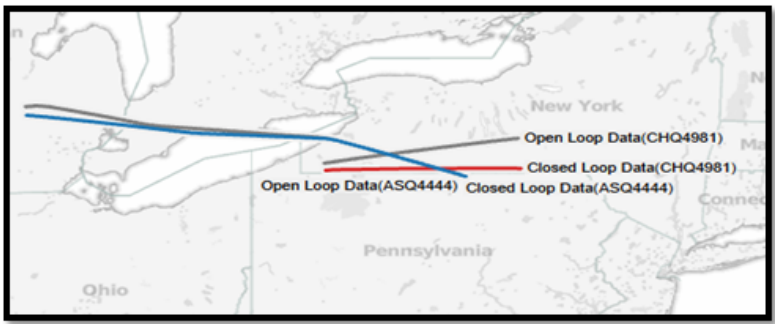
(l) EGF3547-AWE1839



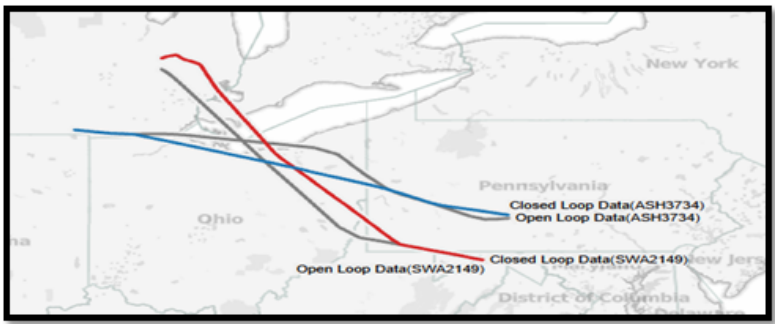
(m) ASQ5139-AWI4081



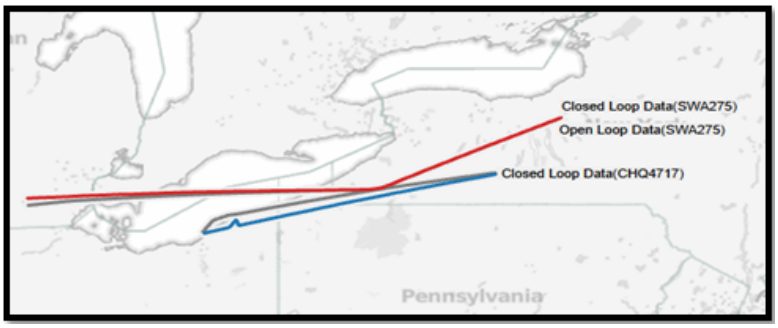
(n) ASQ4640-UAL1527



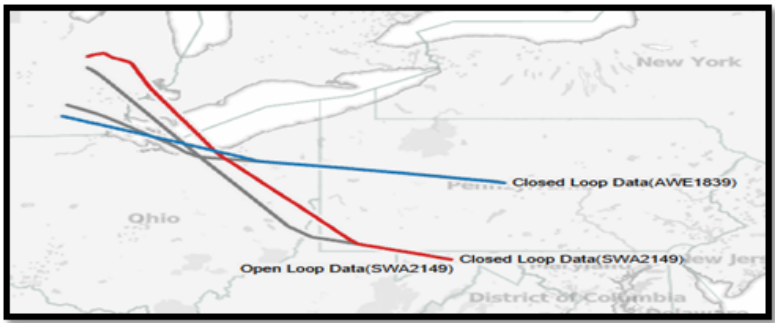
(o) CHQ4981-ASQ4444



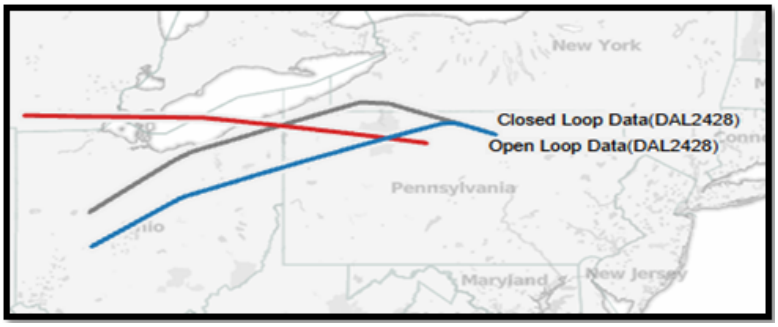
(p) ASH3734-SWA2149



(q) SWA275-CHQ4717

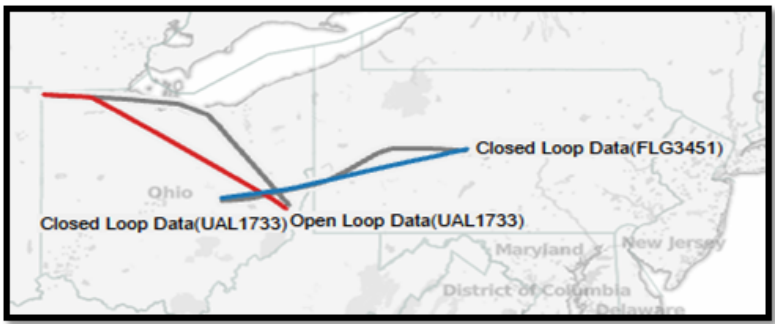


(r) SWA2149-AWE1839

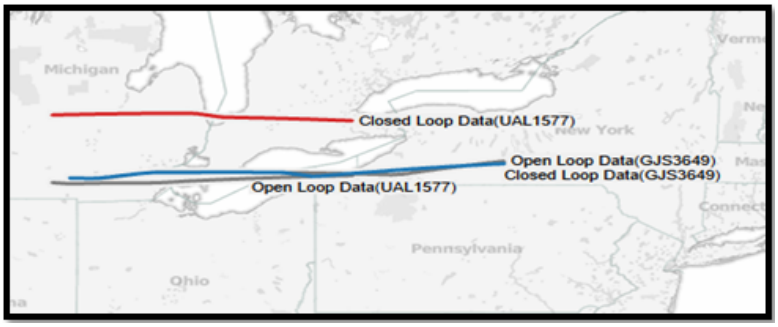


(s) DAL2428-UAL1111

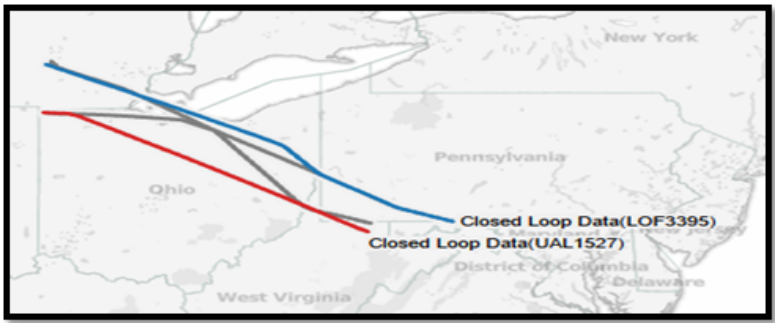
(t) VRD365-EGF2758



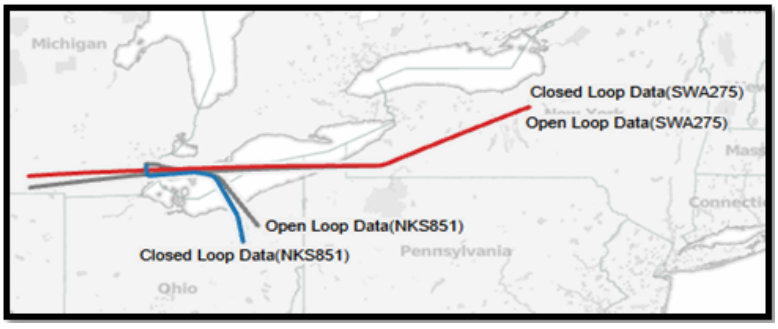
(u) UAL1733-FLG3451



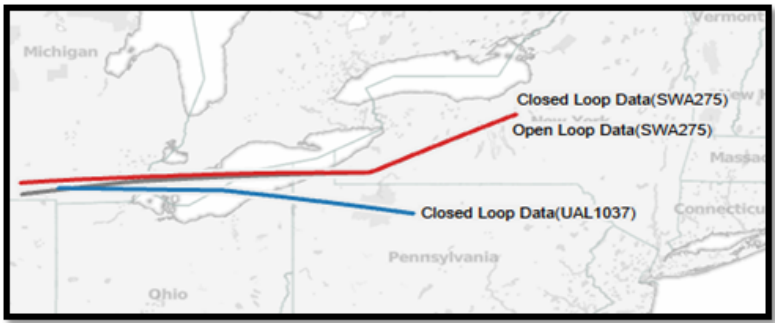
(v) UAL1577-GJS3649



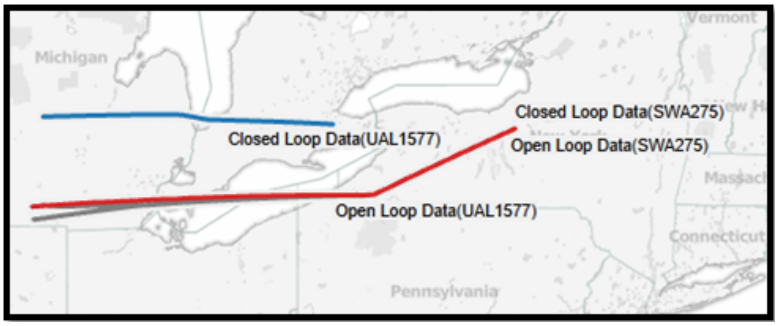
(w) LOF3395-UAL1527



(x) NKS851-SWA275

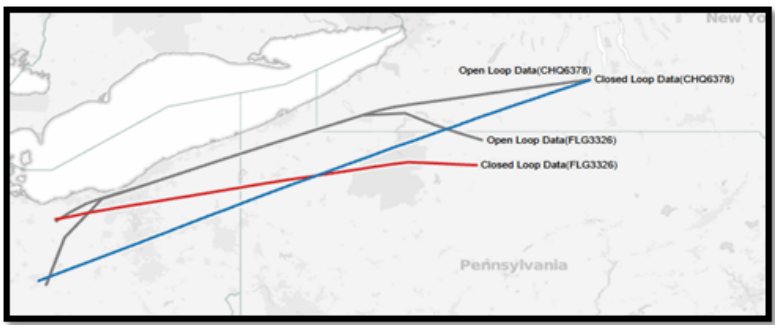


(y) UAL1037-SWA275

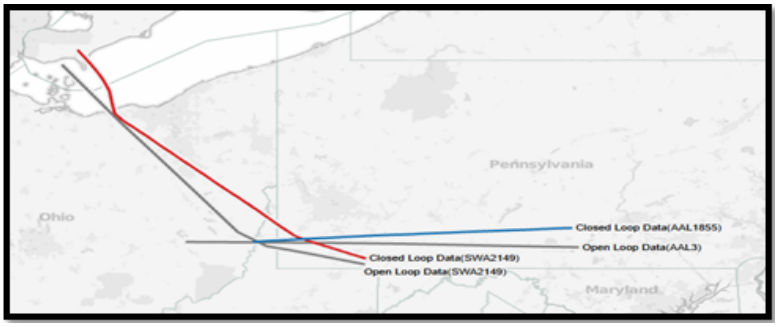


(z) SWA275-UAL1577

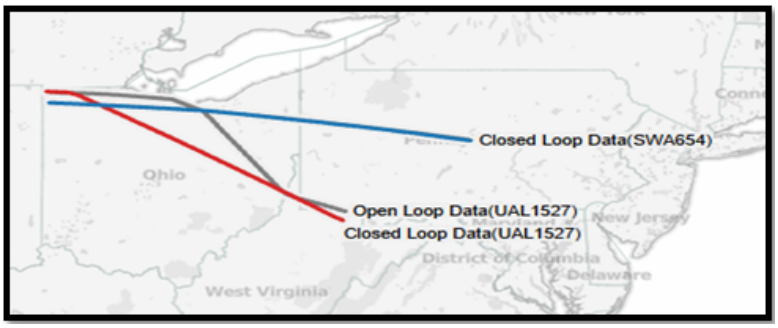
Figure B.2. Aircraft trajectories enountering in conflict(1/2).



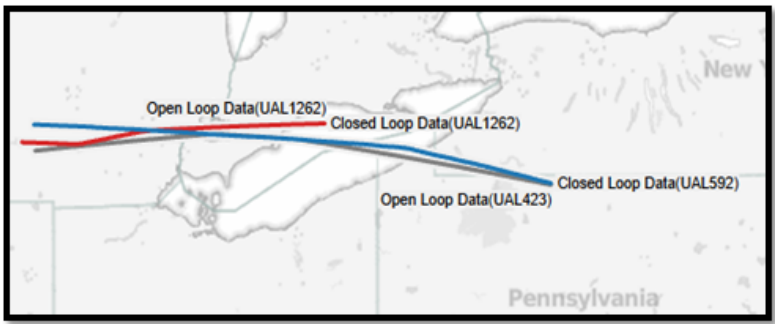
(a) FLG3326-CHQ6378



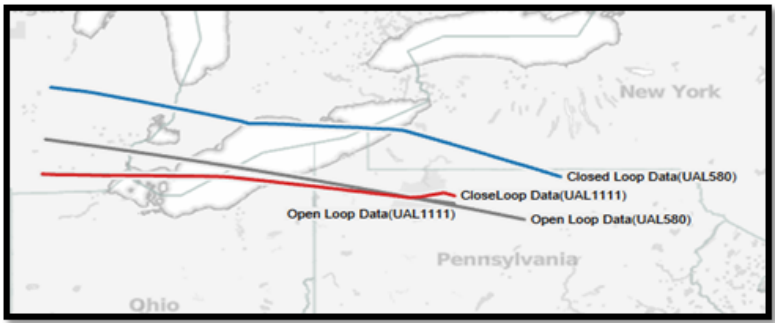
(b) AAL3-AAL1855



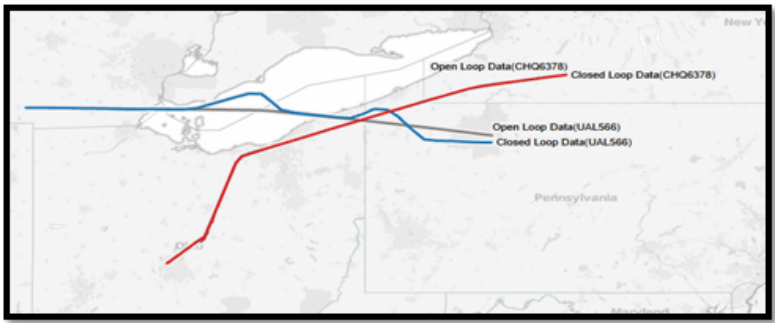
(c) UAL1527-SWA654



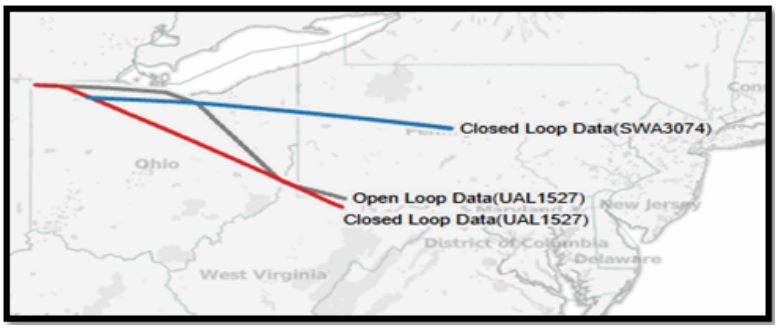
(d) UAL423-UAL1262



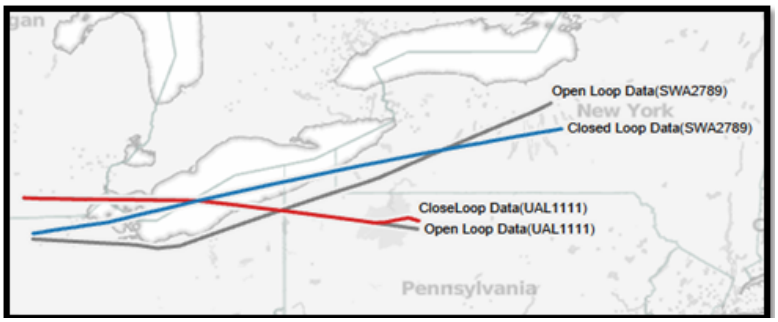
(e) UAL580-UAL1111



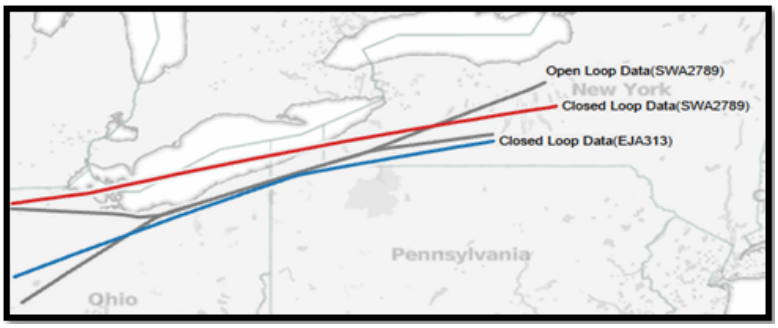
(f) UAL566-CHQ6378



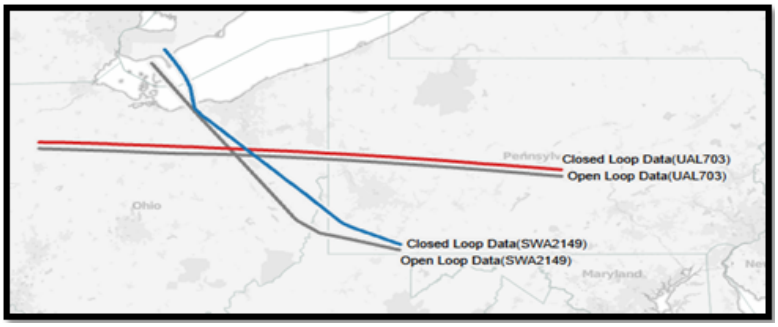
(g) SWA3074-UAL1527



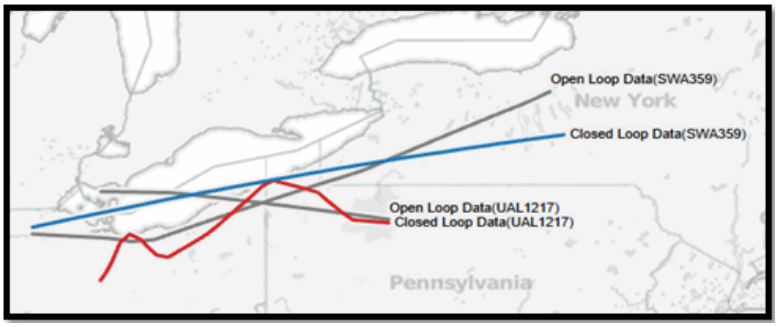
(h) UAL1111-SWA2789



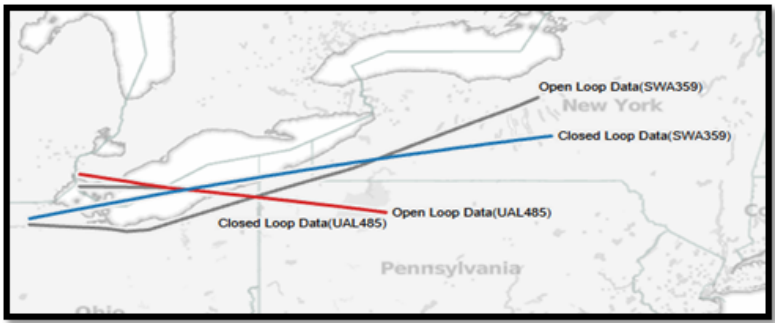
(i) EJA313703-SWA2789



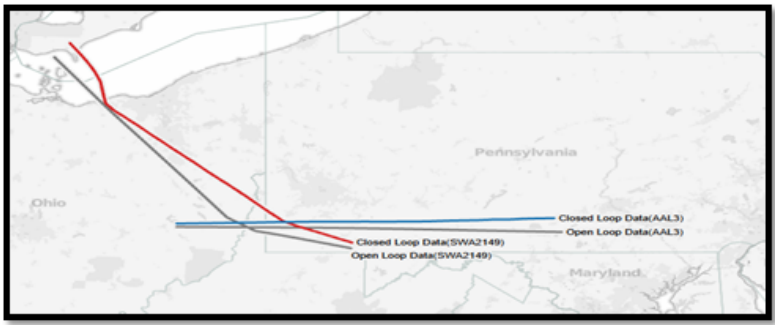
(j) UAL703-SWA2149



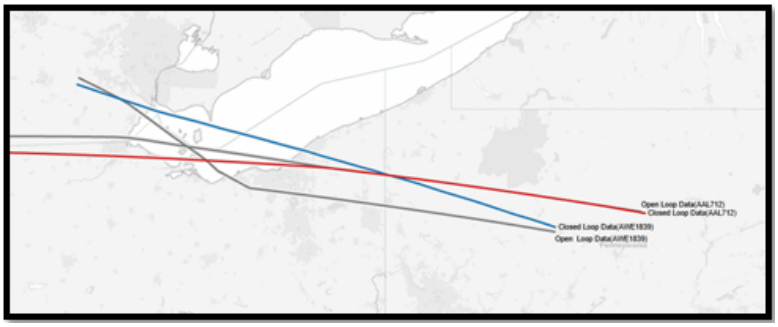
(k) UAL1217-SWA359



(l) SWA359-UAL485

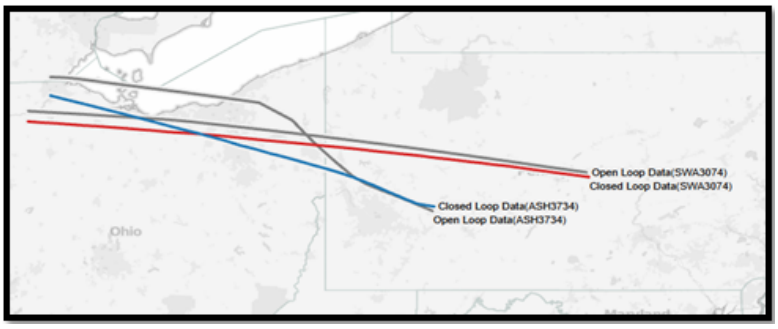


(m) SWA2149-AAL3

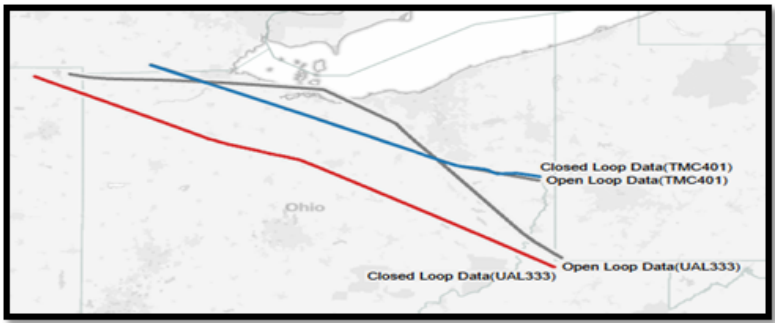


(n) AWE1839-AAL712

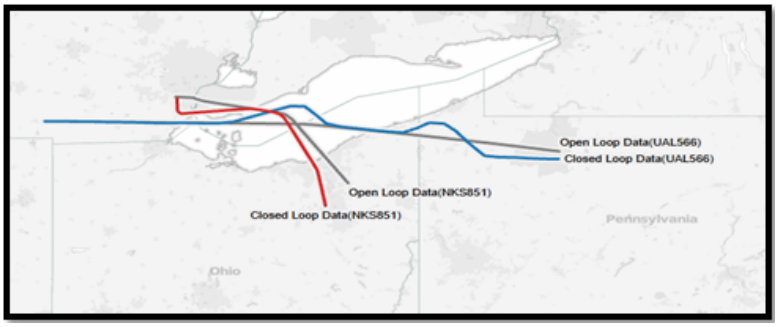
(o) UAL1733-AAL3



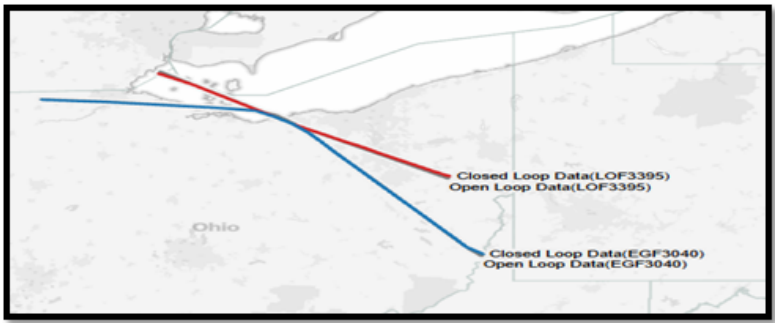
(p) SWA3074-ASH3734



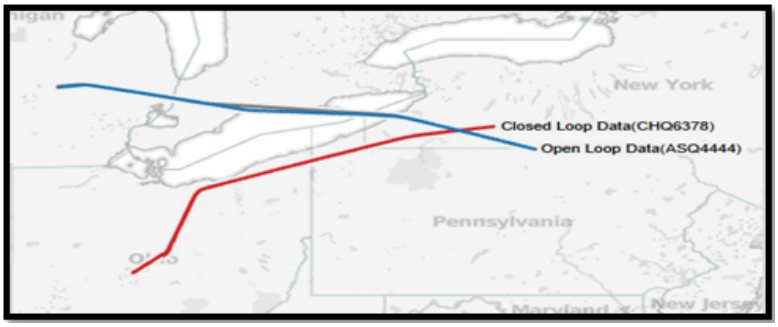
(q) UAL333-TMC401



(r) NKS851-UAL566

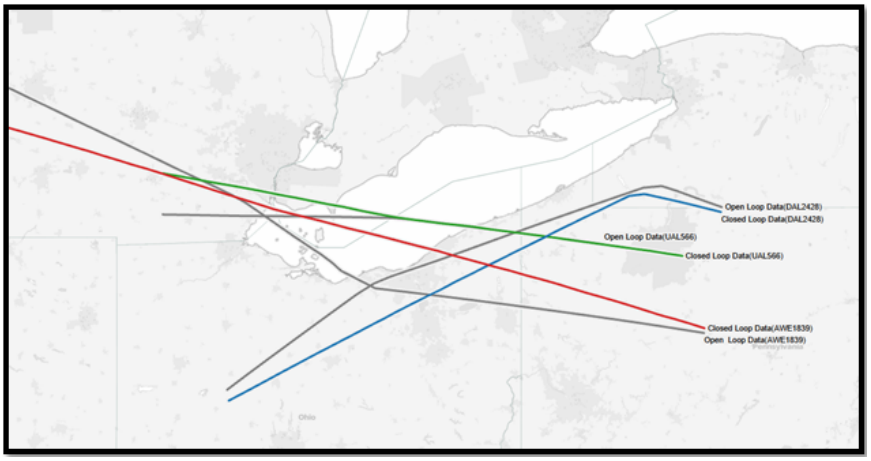


(s) LOF3395-EGF3040(Vertical Dev.)

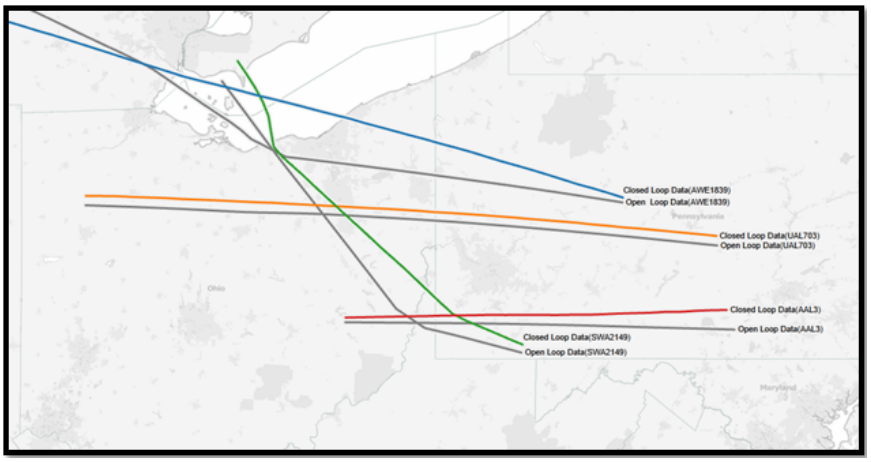


(t) ASQ4444-CHQ6378(Vertical Dev.)

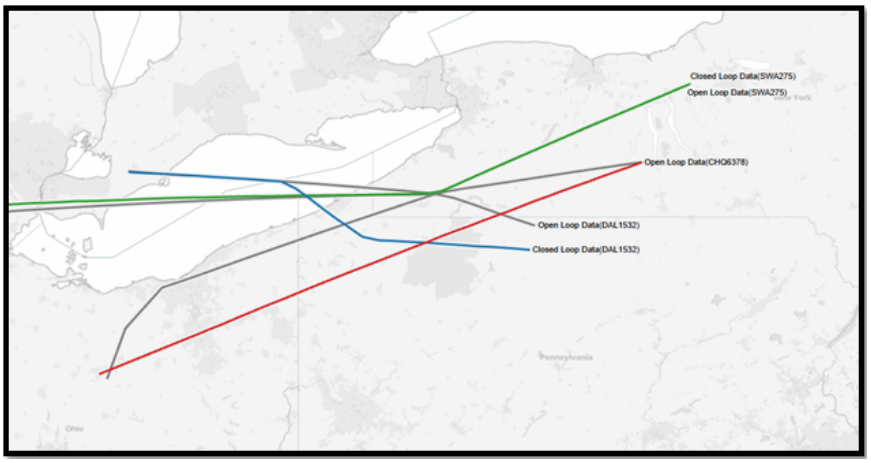
Figure B.3. Aircraft trajectories enountering in conflict(2/2).



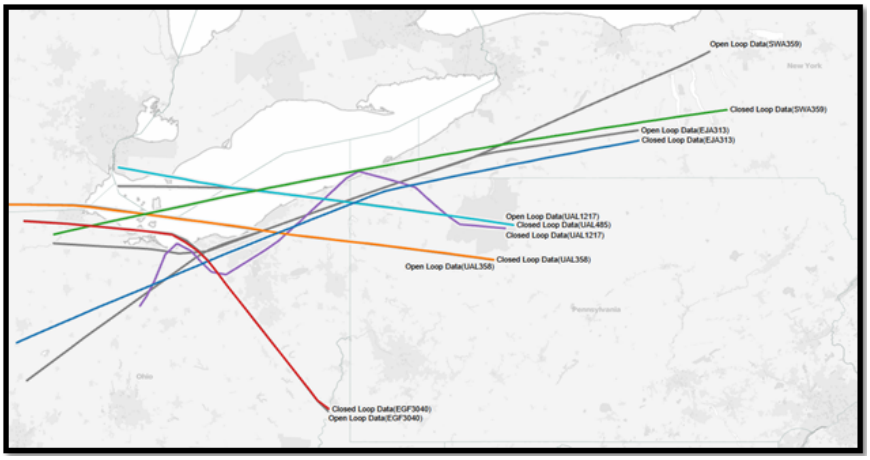
(a) UAL566



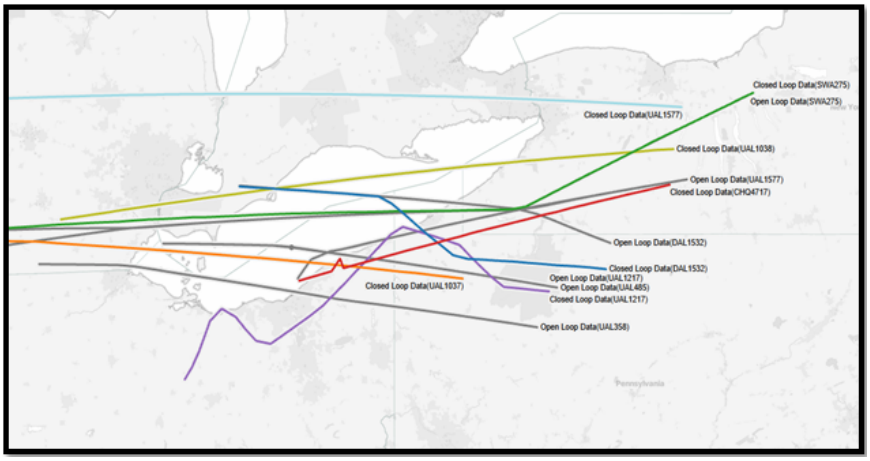
(b) SWA2149



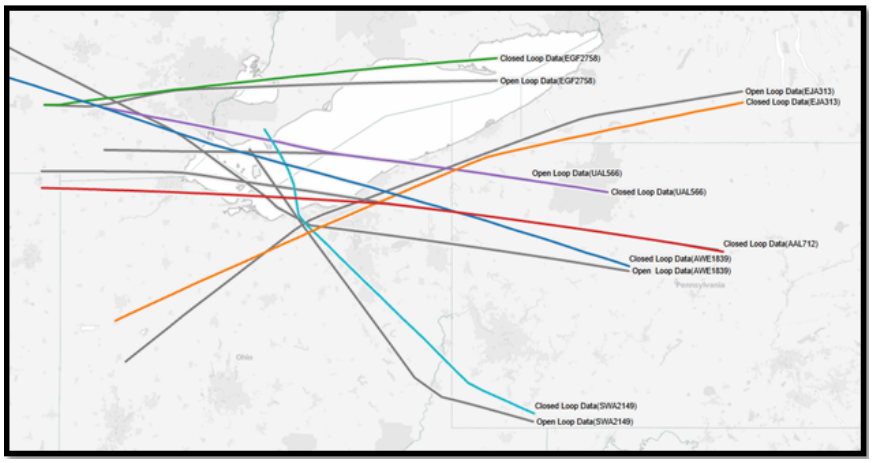
(c) DAL1532



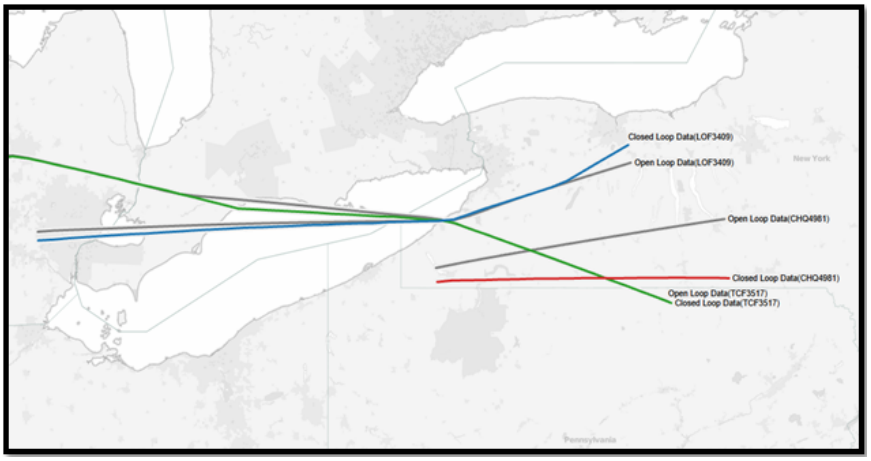
(d) SWA359



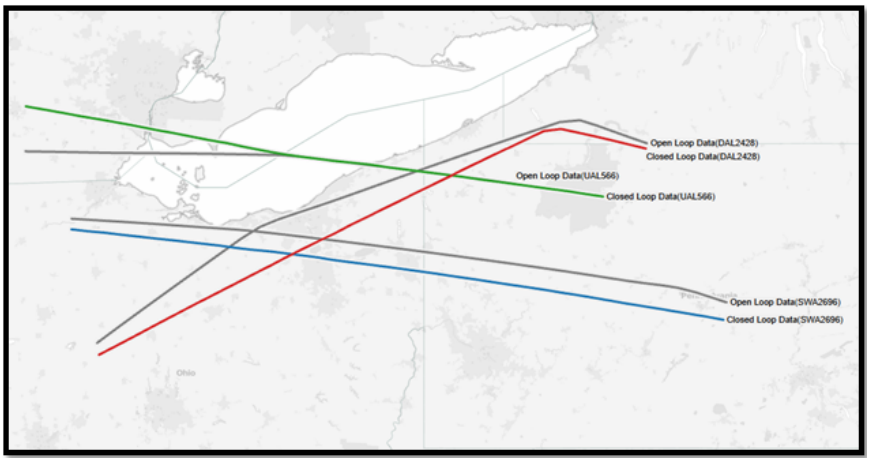
(e) SWA275



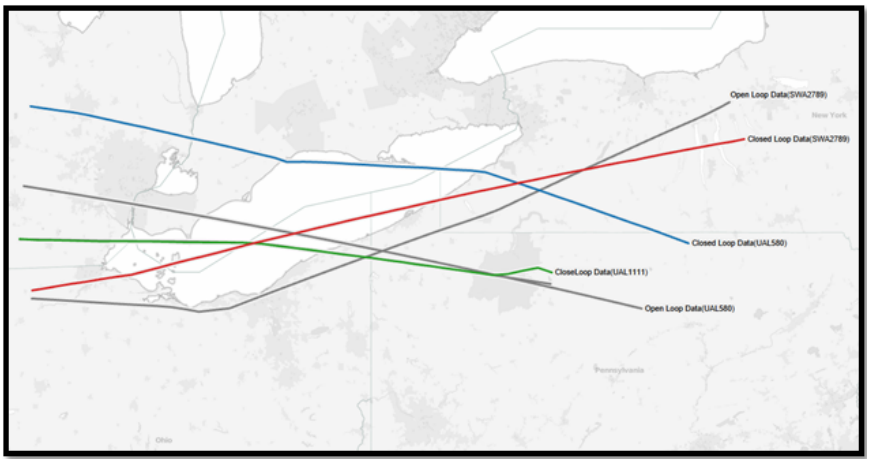
(f) AWE1839



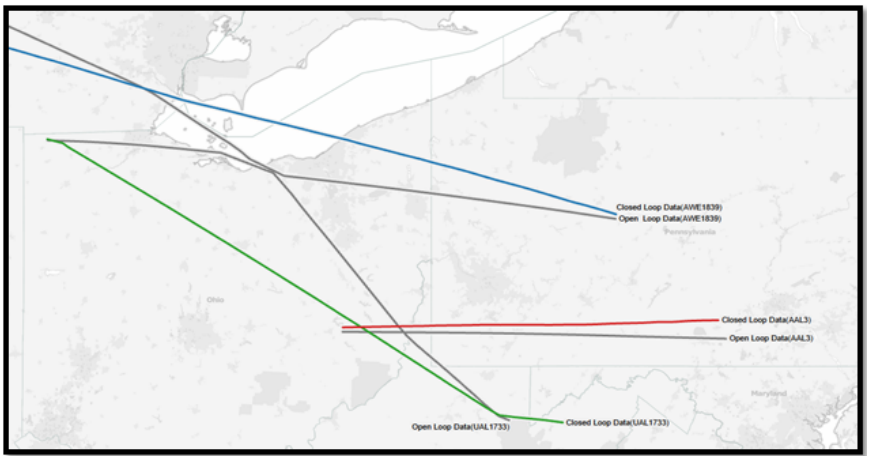
(g) TCF3517



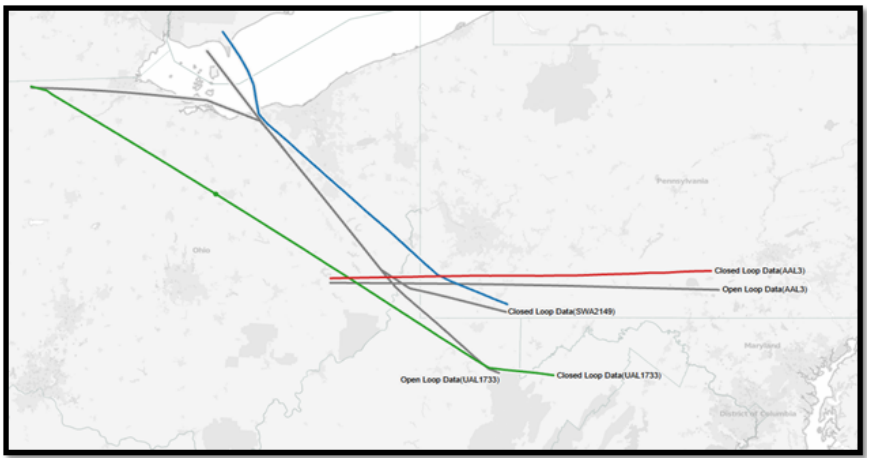
(h) DAL2428



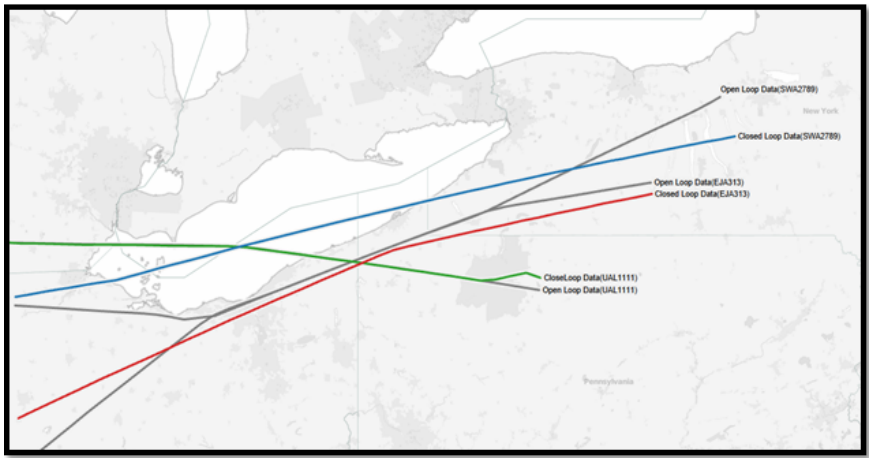
(i) UAL1111



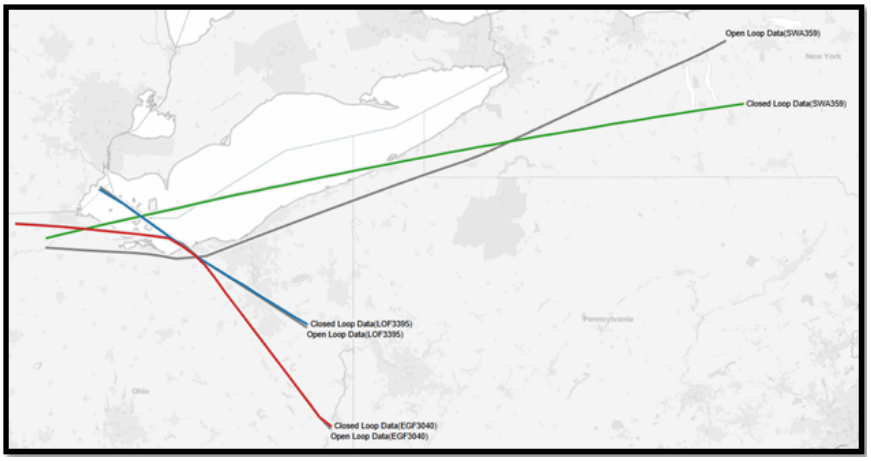
(j) UAL1733



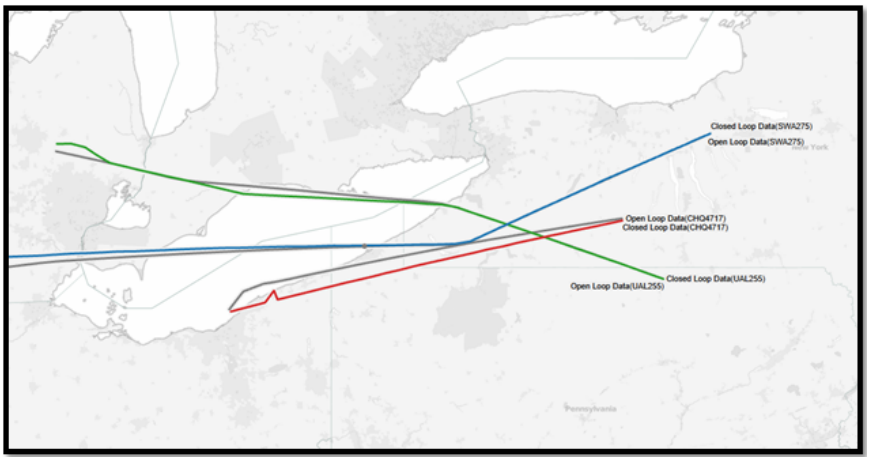
(k) AAL3



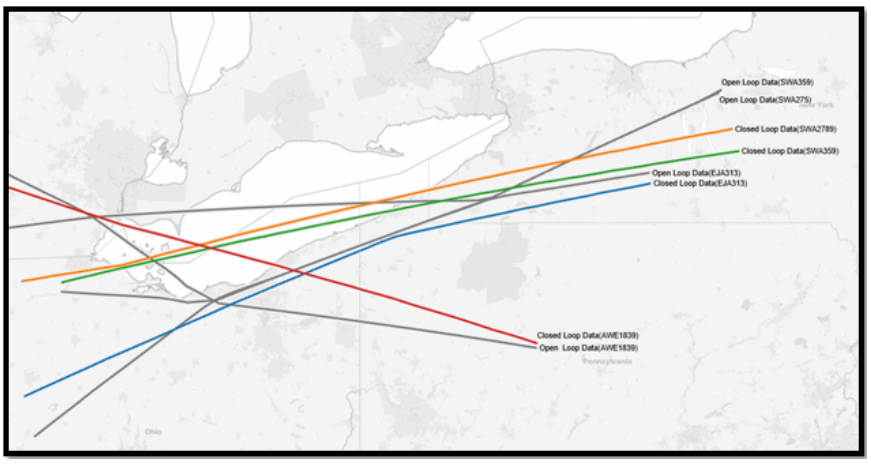
(l) SWA2789



(m) EGF3040



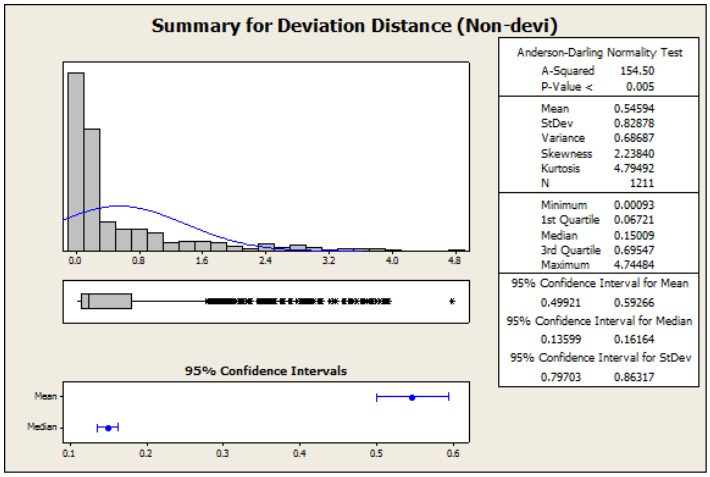
(n) CHQ4717



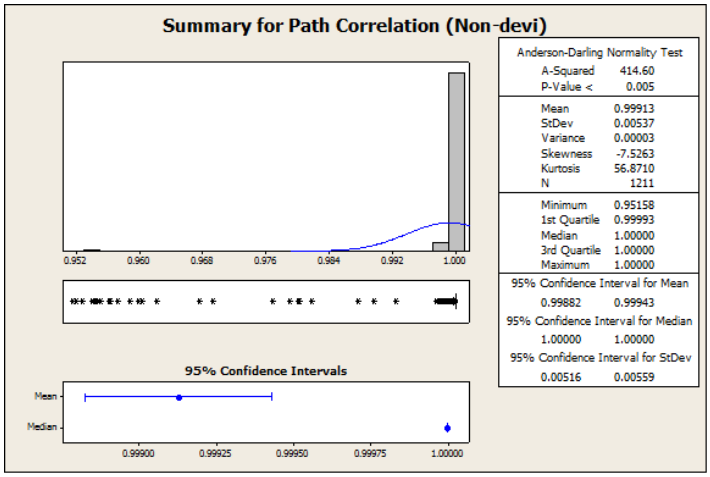
(o) EJA313

Figure B.4. Aircraft trajectories enountering in multiple conflict.

APPENDIX C
STATISTICAL SUMMARY OF DEVIATION CATEGORIZATION

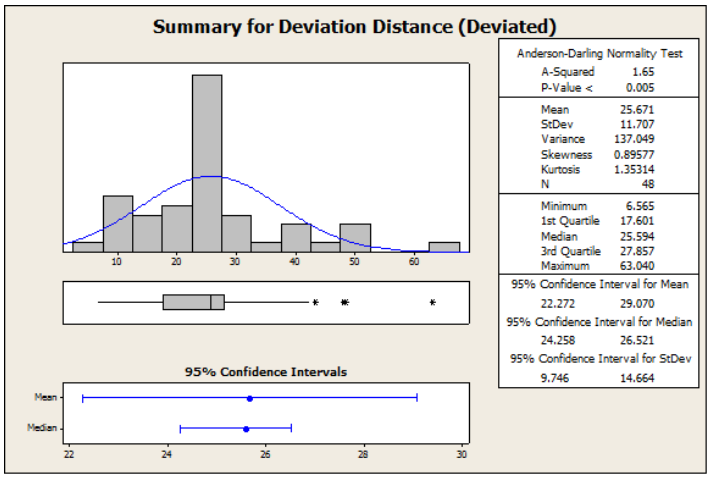


(a) Statistics of "non-deviated trajectory (distance)."

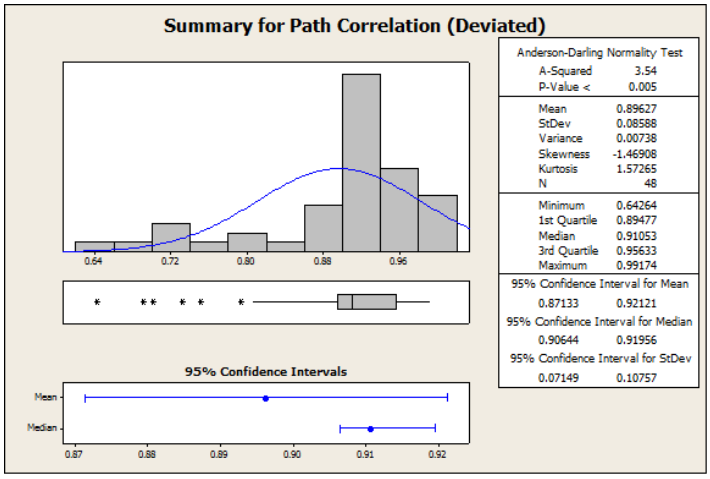


(b) Statistics of "non-deviated trajectory (angle)."

Figure C.1. Statistical summary of "non-deviated trajectory."



(a) Statistics of "deviated trajectory (distance)."



(b) Statistics of "deviated trajectory (angle)."

Figure C.2. Statistical summary of "deviated trajectory"

APPENDIX D
INITIATION TIME FOR HORIZONTAL RESOLUTION MANEUVERS

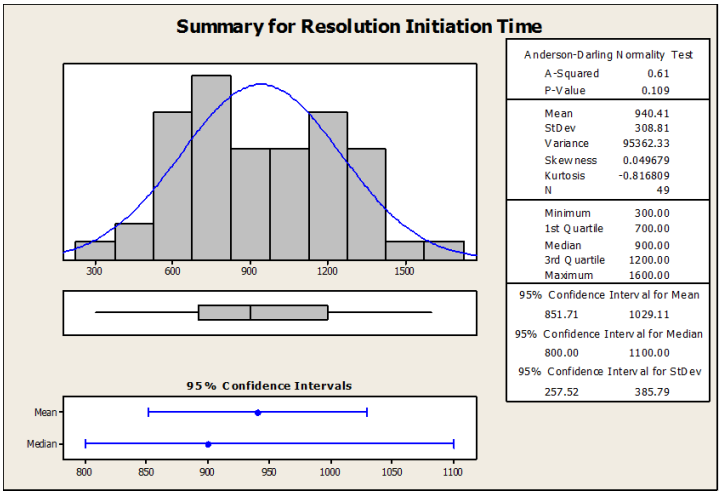
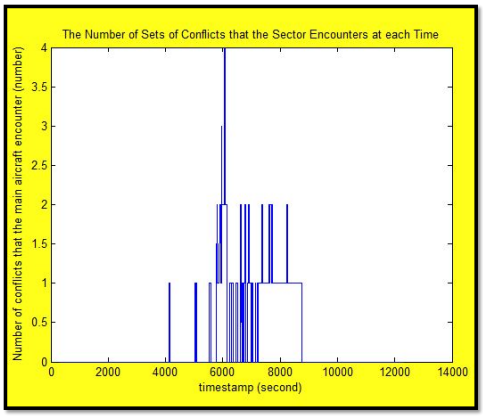
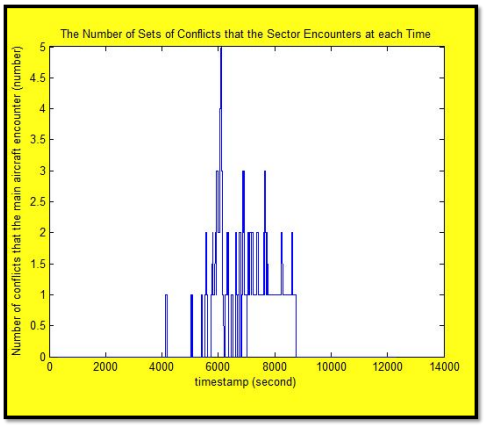


Figure D.1. Initiation time for horizontal resolution maneuvers.

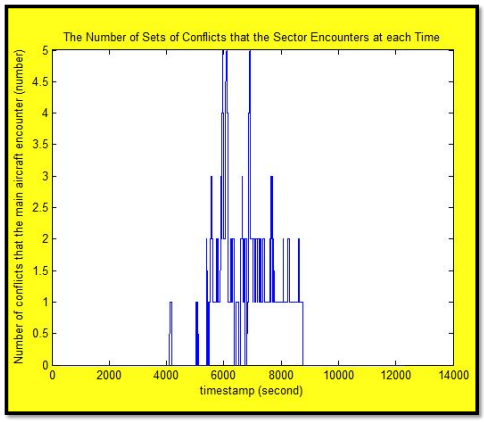
APPENDIX E
CONFLICT DETECTION WITH INCREASED HORIZONTAL RANGE OF PROTECTION ZONE



(a) Protection zone with "6 NM" range.



(b) Protection zone with "7 NM" range.



(c) Protection zone with "8 NM" range.

Figure E.1. Conflict detection with increased horizontal range of protection zone.